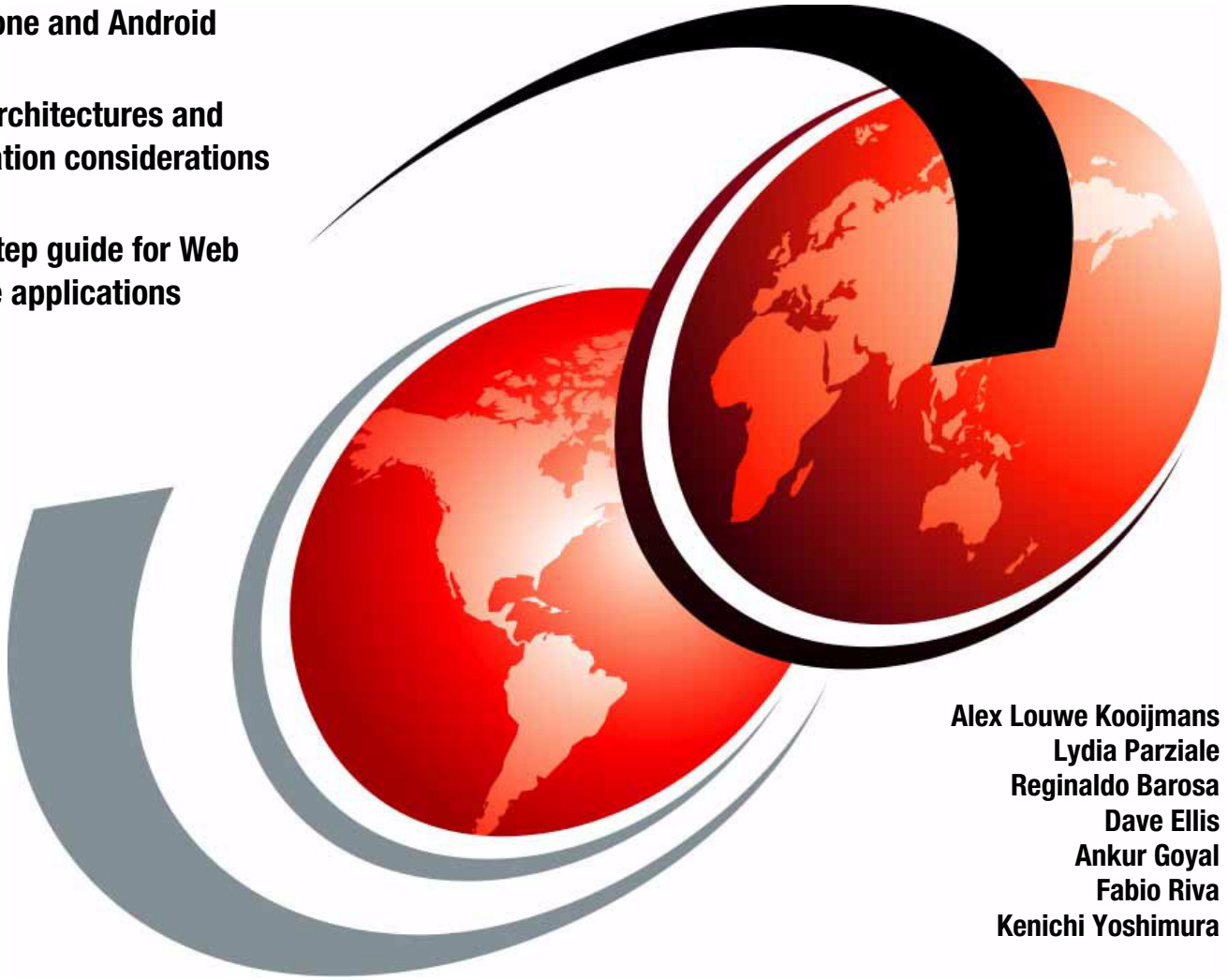
IBM

System z on the Go
Access to z/OS from Smartphones

- Sample scenarios to get started with Apple iPhone and Android
- Solution architectures and modernization considerations
- Step-by-step guide for Web and native applications



Alex Louwe Kooijmans
Lydia Parziale
Reginaldo Barosa
Dave Ellis
Ankur Goyal
Fabio Riva
Kenichi Yoshimura

Redbooks

ibm.com/redbooks

International Technical Support Organization

System z on the Go: Access to z/OS from Smartphones

May 2010

SG24-7836-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (May 2010)

This edition applies to the following IBM software products:

- CICS Transaction Server Version 3.2 and Version 4.1
- WebSphere Application Server for z/OS Version 7
- Rational Developer for System z Version 7.6
- Rational Business Developer Version 7.5
- Host Access Transformation Services Version 7.5.1

© Copyright International Business Machines Corporation 2010. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2010. All rights reserved.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2010. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®
DB2®
developerWorks®
Everyplace®
IBM®
IMS™
Lotus®
MVS™
RACF®
Rational®
Redbooks®
Redbooks (logo)  ®
System i®
System z®
Tivoli®
WebSphere®
z/OS®
z10™
zSeries®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In recent years, many of us believed there was a strong tendency to the substitution of the mainframe systems with departmental ones (sometimes known as ”distributed computing”).

Today we can say this tendency is interrupted, or better we can say that the signals about inversion of the tendency are stronger and stronger. The efforts that have been done in improving the mainframe platform (such as the ability to run open software, openness to the Unix world, integration with not-owning networks and data) has no equivalent in the field of computer science. Despite this big effort and despite the commitments and big investments to renew the platform, some still consider mainframes as old-fashioned or out-of-date. This is not true.

In this IBM® Redbooks® publication we demonstrate that it is possible to combine the traditional strengths of the mainframe to manage large volumes of data and run business transactions with the Web 2.0 paradigm. We can get simpler interfaces, better integration among different services, lightweight protocols for communication, and much more, together with the availability, security, and reliability of mainframe data. And we will show how mainframe data can be accessed by smartphones such as Android or iPhone.

But we can do more to demonstrate how flexible the mainframe platform is. Through the use of pervasive devices it is possible to add new possibilities to mainframe applications, extending System z® capabilities. We can receive notifications in real time, for example, of successful or unsuccessful termination of a TWS job stream, or we can immediately get alerts about abends that occurred in a critical application.

This book is another demonstration that the mainframe is alive and kicking and can and should play a key role in modern application architectures.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Alex Louwe Kooijmans is a project leader with the International Technical Support Organization (ITSO) in Poughkeepsie, NY. He specializes in SOA technology and solutions on System z. He also specializes in application modernization and transformation on z/OS®. Previously he worked as a Client IT Architect in the Financial Services sector with IBM in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand services. Alex has also worked at the Technical Marketing Competence Center for zSeries® and Linux® in Boeblingen, Germany, providing support to customers getting started with Java™ and WebSphere® on System z. From 1997 to 2000, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks projects and delivering workshops around the world in the areas of WebSphere, Java, and e-business technology on System z. Before 1997 Alex held a variety of positions in application design and development, product support and project management, mostly in relation to the IBM mainframe.

Lydia Parziale is a project leader for the ITSO team in Poughkeepsie, New York, with domestic and international experience in technology management, including software development, project leadership, and strategic planning. Her areas of expertise include

© Copyright IBM Corp. 2010. All rights reserved.

e-business development and database management technologies. Lydia is a certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management. She has been employed by IBM for over 20 years in various technology areas.

Reginaldo Barosa is an IBM Senior Certified Application Development Specialist. He provides technical support, helping customers with enterprise modernization and development tools. Before joining IBM US ten years ago, Reginaldo worked for 27 years in IBM Brazil. He has co-authored IBM Redbooks and has written two books, as well as other articles and tutorials for IBM developerWorks®. He holds a degree in Electrical Engineering from Instituto Mauá de Tecnologia, São Paulo, Brazil. You can reach Reginaldo at rbarosa@us.ibm.com.

Dave Ellis has worked with IBM mainframes for 34 years with jobs ranging from being an operator through application programming to CICS®, z/OS and IMS™ systems programming in a variety of locations throughout the UK and the Middle East. In the mid 1990s he moved into software development with the Omegamon range of products and now currently develops product code for the IBM Tivoli® Omegamon XE for Storage on z/OS products. In addition, over the past few years he has worked on utilizing Web 2.0 technologies to access traditional mainframe systems programming data and functionality, in particular from mobile devices.

Ankur Goyal is an IT Specialist from IBM working with the Rational® team in India. He consults customer solutions on IBM middleware, open source, open standards, and emerging technologies such as enterprise modernization. His areas of interest include software development best practices and integrating Web technologies with the help of open standards. He joined IBM in 2004 and holds a Bachelor's degree in information technology from National Institute of Technology, Durgapur. He is IBM certified for J2EE, DB2®, WebSphere, Rational, XML, and SOA.

Fabio Riva is a Senior IT Architect in Technical zSoftware Presales, SWG IBM Italy. Joining IBM in 1985, he held various positions inside the company, from MVS™ System Programmer to Systems Engineer, up to Senior IT Architect. In recent years his main activities were related to revitalization of mainframe market and business development on System z. Fabio was involved with the development of cross-brand and cross-platform (hybrid) solutions, with the main focus on the mainframe platform. He is also acting as a technical presales agent for some Tivoli products in the areas of SW asset management, licence management, cost management, and security on System z. Fabio authored an IBM Redbooks publication about architecting WebSphere on z/OS. He published several technical articles and publications inside IBM, but also articles in external newspapers (such as Computerworld). He was also a speaker at several international conferences and technical user groups.

Kenichi Yoshimura is Development Manager in the IBM Australia Development Laboratory, and is responsible for the development and service of the File Manager for z/OS product. Since joining IBM, he has worked on a number of projects including Fault Analyzer for z/OS and SCLM Developer Toolkit.

Thanks to the following people for their contributions to this project:

Dennis Weiand
IBM Sales & Distribution, TSS
Certified SW IT Specialist - Software - CICS, Web & Java - Advanced Technical Skills (ATS), Americas

Nico Chillemi
IBM Sales & Distribution, Software Sales

Executive IT Specialist - zChampion

Francesco Carteri
IBM Software Group, Tivoli
Software Engineer Software Developer

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks
- Send your comments in an e-mail to:

  redbooks@us.ibm.com
- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html

1

Introduction

Mainframe computers have changed dramatically over the years. Starting from the old monolithic, centralized, proprietary machines of the 60's, now we have an open server, ready to support non-proprietary protocols, capable to run open software and fully integrated with all the other distributed platforms.

Mainframes are now able to consolidate distributed computing. They are still capable of executing "traditional" mainframe workloads, but at the same time, they also have the ability to run new workloads on the same machine. This quality of the mainframe platform, and its openness, are universally recognized.

However, even though a lot of work has been done in improving the mainframe platform, in the minds of many people, mainframes are still legacy machines typically accessed with 3270 terminals ("green screen" terminals). The reality has been different for many years now, but the old mindset is still alive. Much of this misperception comes from the bad feeling of missing appealing access capabilities. System z supports several alternatives to the 3270 interface; and almost all software products can work with TCP/IP-based access and can be used in the same way as on distributed platforms. Our intention with this IBM Redbooks publication, therefore, is to demonstrate how easy it is to add capabilities that not everyone considers possible with the IBM mainframe.

© Copyright IBM Corp. 2010. All rights reserved.

1.1 Web access, Web 2.0 and mainframes

Many technical people or data-entry users still use a 3270 terminal emulator to access the mainframe as a matter of tradition. They have worked that way for years. Others prefer the 3270 terminal emulator because they feel it is the best method to access data in terms of performance. Even when there is no possibility to convince purists or headstrong people to change their minds, we must take into consideration the fact that the World Wide Web is now the standard for accessing information. Use of the Web browser has become the standard way to access information because it is simple, standardized, immediate, and easy to learn.

So, the use of the Web to obtain data from mainframes can be the right solution to simplify System z access. But we can do more. There is another emerging technology that can be used to access mainframes: *Web 2.0*. Web 2.0 offers even more simplified Web access.

The traditional Web platform (sometimes called *Web 1.0*) was often diminished due to the static nature of the data and its limited functionality, if compared with typical desktop applications. Web 2.0 can add desktop application-like functionality, using asynchronous JavaScript and XML (AJAX), and other enabling technologies such as REpresentational State Transfer (REST) or Atom.

Flexibility is the key driver of Web 2.0 success. Web 2.0 enables the flexible delivery of data through the combination of services and disparate data sources through mash-ups, real time data feeds, and rich interactions. We describe all of these in more detail later in this book.

There is also another aspect to consider. Web 2.0 can be nicely integrated with other strategic computing directions, such as service-oriented architecture (SOA). Enabling Web 2.0 functionality requires infrastructure flexibility, which can be supported by a robust SOA. Both AJAX and REST are enablers for SOA. For example, REST can be used to expose services, and AJAX is used to build front ends.

If you add this to the mainframe performance, data management, reliability, and application strength, you will be able to build a very powerful solution for any IT enterprise.

1.2 What Web 2.0 is

Web 2.0 offers more than simple Web access. As we said before, it is composed of many enabling technologies, allowing users to access data in a more innovative way. With Web 2.0 there is a change in how users receive information: from the passive receiving of information provided by others (as it was in Web 1.0) to active contribution in writing content.

Users are no longer only the “receivers” of information, but have an active role as contributors of information. Web 2.0 means a new business model designed on user need.

The main technology designed to change user access is called AJAX, which allows users to interact with Web applications having similar functionality as desktop applications: context-sensitive menus, drag-and-drop, right mouse click, and information updates without a complete screen refresh. An AJAX interface provides the same “look and feel” as desktop applications, but with all the benefits of Web access. This means data and information can be received from everywhere, no software to install and maintain, centralized data (and the data can stay on the mainframe, exploiting its traditional strength).

But what does the Web 2.0 paradigm mean in concrete terms? Web 2.0 is not rocket science. It is simply a very intuitive, fast and customizable user interface, joined to high performance (yet easy to use) software products and tools.

1.2.1 Easy access to services

Web 2.0 applications should be *simple to access* because the services are accessed via a simple Web API interface (typically using REST with AJAX-capable browsers), and *simple to use* because the same services are exposed on the enterprise with the use of Internet addresses and feeds (generally Atom feeds). But what do these acronyms mean?

AJAX

One of the technologies used in Web 2.0 is AJAX. The acronym means Asynchronous JavaScript And XML. Figure 1-1 shows both the flow of a traditional connection and an AJAX connection.

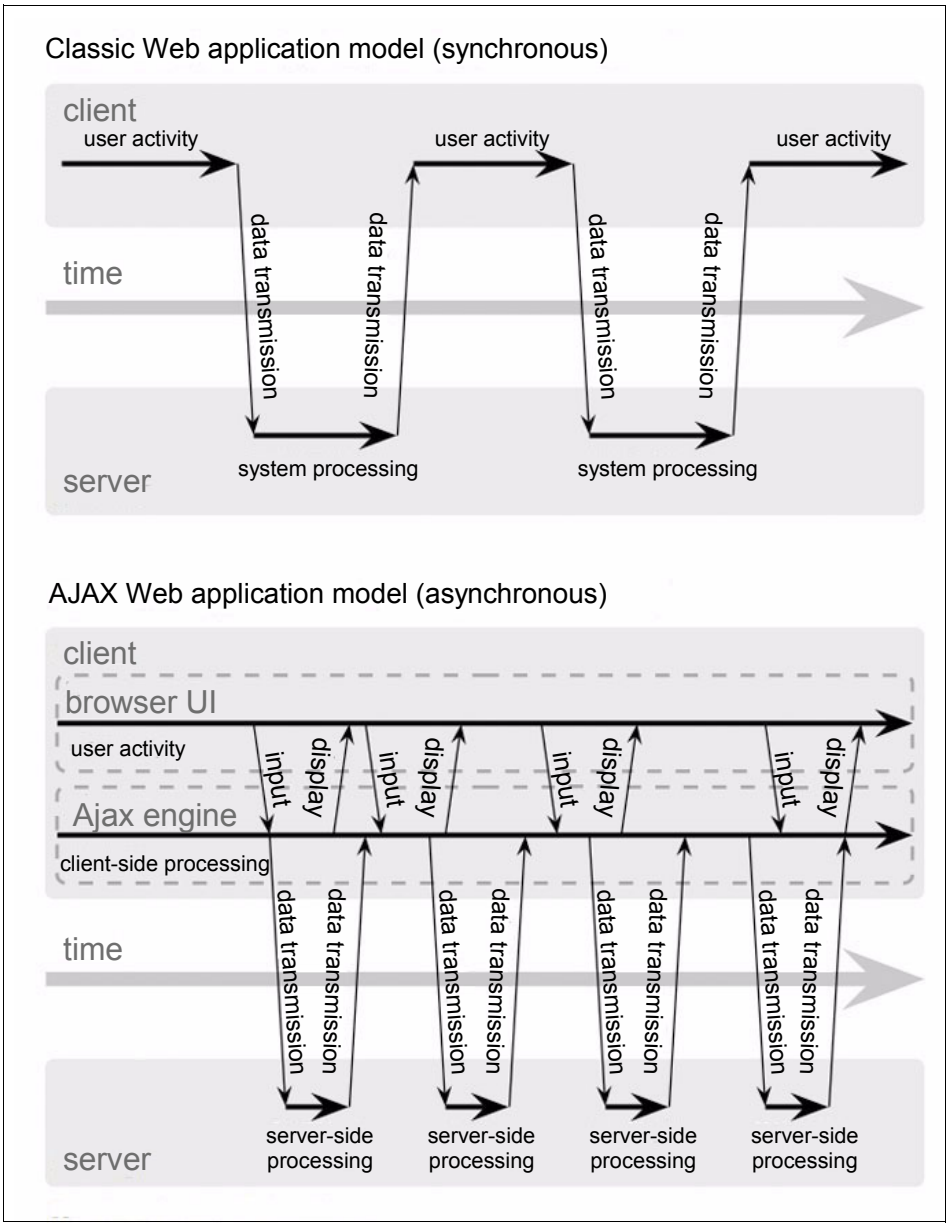


Figure 1-1 AJAX flow

This technology allows users to access data (from any platform, including the mainframe) in an asynchronous way to reduce user wait times. AJAX is not really new, but is a combination

of existing technologies: XMLHttpRequest, JavaScript, CSS, and XML. AJAX is asynchronous. It allows you to update only parts of a Web page so you no longer need to refresh the whole page when you want the user to interact with the server.

REST

Another technology that is part of Web 2.0 is REpresentational State Transfer, or REST. It is a style of software architecture for distributed hypermedia systems such as the World Wide Web. It is a lightweight approach to interconnecting a front end with the public Internet. It uses HTTP and XML.

REST makes it easier to access data on a server because it uses a set of well-known, standard operations such as GET, POST, PUT, and DELETE. Using these, it is possible to interact with a piece of data, called a resource. A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet that identifies the resource, and you can get its value using the GET operation, update it using the POST operation, and so on.

A RESTful Web service is formed like a sentence: it simplifies how developers access services. As seen in Figure 1-2, the VERB would be equal to the HTTP action (GET, POST, PUT, DELETE), the noun would be the URI of the service (the document) and the adjective would be the MIME type of the resulting document.

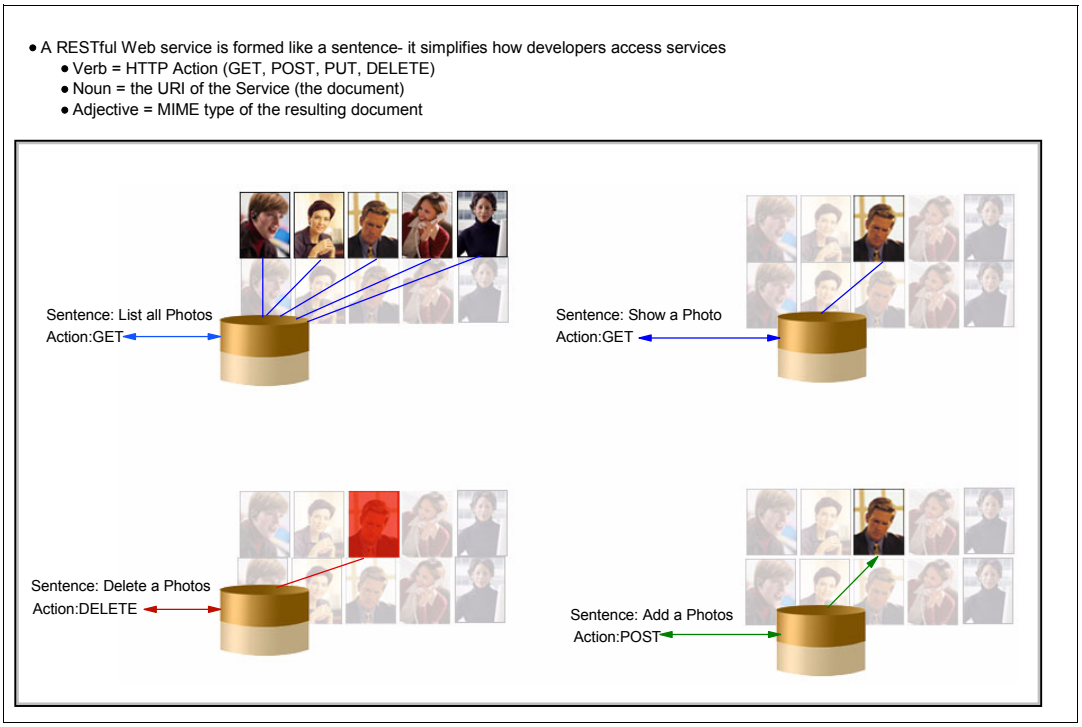


Figure 1-2 REST: operations on resources

Atom

The Atom Syndication Format is an XML language used for Web feeds. A *Web feed* is a data format used for providing users with frequently updated content. Atom is an evolution of RSS Web feed format that is more structured. It was developed to solve some RSS limitations such as not allowing well-formed XML markup in its content.

1.2.2 Reuse of existing services (RESTful SOA)

Using the Web 2.0 technology, it is possible to make existing services at the enterprise level more accessible. More and more IBM products allow users to access data via the REST protocol. REST has become perhaps the single most widely deployed form of service orientation because of its simplicity.

The merger of lightweight Web 2.0 technologies with the robust SOA infrastructure is called *RESTful SOA*. Key aspects of building an effective RESTful SOA are:

- Take advantage of your existing infrastructure wherever possible.
- Use well-established, ubiquitous technologies for scalability, performance, and security.
- Build rich user interfaces that run in any browser.
- Make content simple and human readable.

Figure 1-3 shows the relationship between Web 2.0 and SOA. Web 2.0 is nice but should be based on a robust SOA infrastructure in order to build a powerful IT solution.

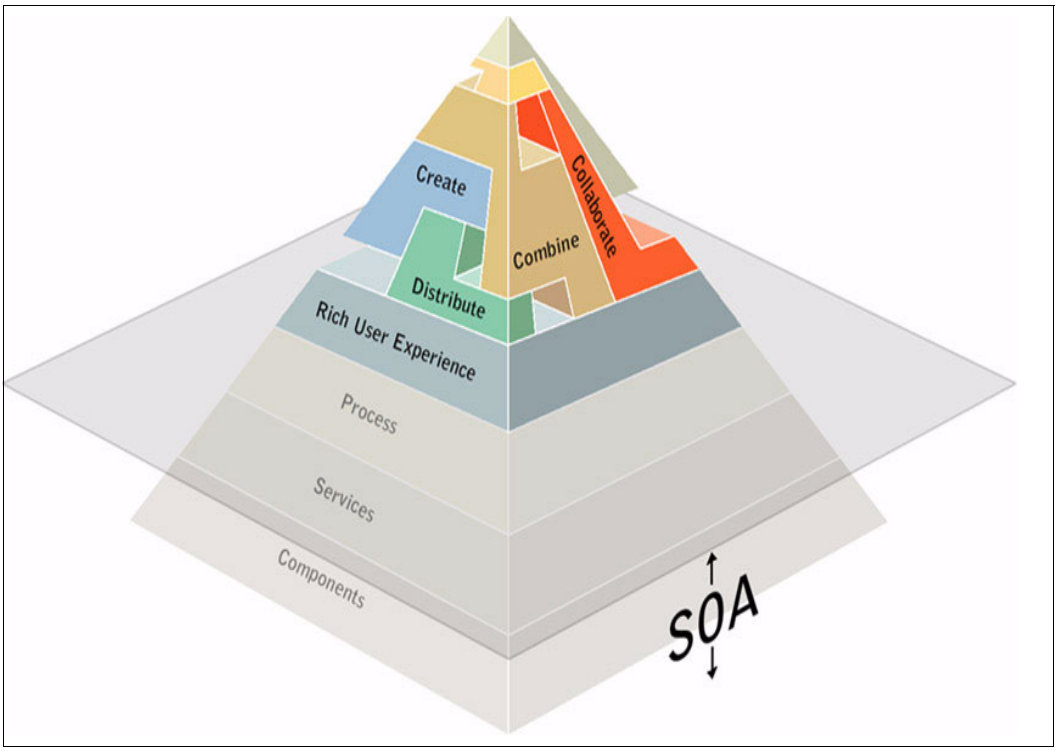


Figure 1-3 SOA and Web 2.0 combined

1.2.3 Mashup among services (internal and/or external)

Together, SOA and Web 2.0 fulfill the promise of flexibility and agility for a business process. Web 2.0 mashups provide the framework to leverage the business blocks to deliver information to people rapidly and flexibly.

In an enterprise context, Web 2.0 enables workers who do not have technical knowledge to build ad-hoc collaborative “enterprise mashup” applications on their own to address immediate business needs. By using readily available and intuitive tools, they can assemble existing content, wire it together, and share it with their coworkers, customers, and business

partners to facilitate business in an exciting new way. We provide some examples of this in this book.

In Web development, a *mashup* is a Web page or application that combines data or functionality from two or more external sources to create a new service.

With the mashup capability, even small and medium businesses can produce strong business functionality at reasonable costs. With the ease of mashup, users might not even need to wait for their information technology group. They will begin to develop their own applications that meet business needs. This brings us to a new concept: the *Long Tail*.

1.2.4 Long Tail and consolidation of situational applications

The term *Long Tail* was first coined by Chris Anderson in an October 2004 Wired magazine article to describe certain business and economic models such as Amazon.com or Netflix. Chris Anderson said that businesses with distribution power can sell a greater volume of hard to find items at small volumes than of popular items at large volumes.

Applied to the IT world, this concept can be translated into the following: in addition to mass-market and enterprise applications, there are also a great number of situational and niche applications developed directly by users. We should take them into consideration, due to their number.

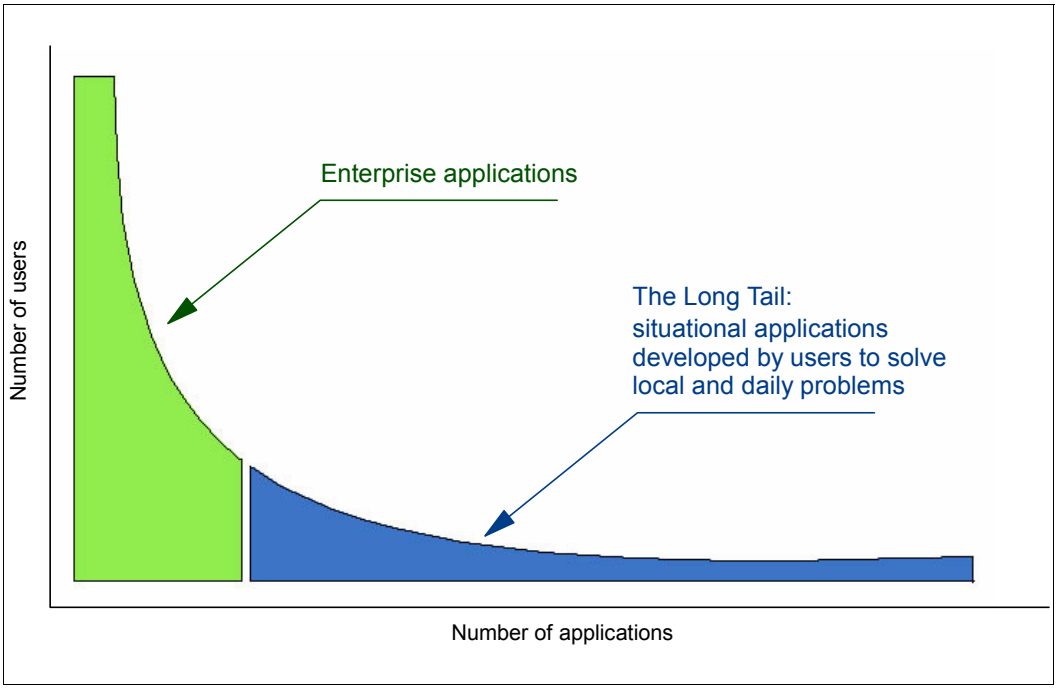


Figure 1-4 Long Tail with Web 2.0

Some of these situational applications are now critical for companies. They were developed by users with low technical skills but with high knowledge of company business. Generally, these applications are mashups of existing services, combining data from several sources.

The strategic direction in this area should be to provide a common platform to develop these kinds of applications. This platform should be open and flexible. In order to be sure that it will be used by users, this platform should be free and easy to implement.

If we use a common platform to develop and execute situational applications, we will be able in the future to consolidate all of them or convert them on another similar platform, but managed by the IT structure for all enterprises. The solution is already in place and is available free of charge from the Internet: it is called *WebSphere sMash*, also known as *Project Zero,* and can be found at:

http://www.projectzero.org

Project Zero is a technology incubator project centered around agile development and the next generation of dynamic Web applications. The project introduces a simple environment for creating, assembling, and running applications based on popular Web technologies. This environment includes a scripting runtime for Groovy and PHP with application programming interfaces optimized for producing REST-style services, integration mashups, and rich Web interfaces.

Consolidation of sMash applications can be realized either by migrating applications to the WebSphere Application Server platform (some conversion activities are involved) or directly consolidating sMash applications on the System z platform, using Linux on System z (starting with WebSphere sMash V1.1.1).

1.3 Extending the mainframe platform: the use of smartphones

A *smartphone* is a mobile phone offering advanced capabilities, often with PC-like functionality. A variety of platforms and operating systems are available including Symbian, RIM Blackberry, Apple iPhone, Windows® Mobile, and Google Android. Smartphones started to appear in the market in the 1990s. However, the real evolution began in early 2000. We saw an early adoption of smartphones predominantly by executives on the go as a fancy alternative to having a laptop to access Web and e-mails from anywhere in the world.

The acceptance of smartphones appears to have accelerated in recent years by the introduction of Apple's iPhone, which has gained acceptance by young and old. Its user interface is a major contributor to the wider acceptance of technology that caters to the needs of novice users as well as technology savvy users.

With the introduction of the Android operating system and its open software development tools and environment in 2007, along with open standards for mobile devices, the development of smartphones and its applications is expected to accelerate in coming years.

Accessibility to cheap, reliable, and fast wireless networks, coupled with the availability of computational power on smartphones, has led to much growth and potential of smartphone applications, particularly in the entertainment industry (for example, online games). We expect to see mainframe-based solutions playing a part in smartphone application scenarios in coming years.

1.3.1 Smartphones open up a new way of doing business

Smartphones are dramatically changing the way people use the Internet. More and more people are using the mobile phone for Internet and intranet access, besides for the core capabilities such as text messaging and making phone calls. The smartphone is starting to behave as a small portable computer, sometimes even with a full suite of office applications, and often with Internet access.

It is obvious that smartphones expand business opportunities, simply because of the fact that people are "connected" to the world of e-commerce more hours each day. By using

smartphones, the power of the Internet is not only reaching users whenever they are at home or at work and physically behind a computer, but also when they are in any other location, such as enjoying a nice sandy beach.

Typical scenarios for a smartphone could be:

- An airline passenger checking in on a flight from his smartphone on his way to the airport.
- An accounting professional checking some numbers in the General Ledger while taking the train home from work on the last day of the quarter.
- A warehouse manager, who is typically spending most of his time walking around in the warehouse, receiving an alert on a smartphone that an item is out of stock and needs replenishment immediately.

But also, a smartphone can help in, for example, day to day operations of a big mainframe: An operator “on call” receiving alerts on his/her smartphone on the status of critical jobs.

1.4 The scenarios in this book

We demonstrate two distinct usage scenarios in this book:

- First, accessing mainframe-based information on demand. That is, users would proactively request information or service and information is provided. IBM offers a wide range of solutions around SOA and Web 2.0 to make mainframe-based resources accessible, but also tools to consume the information efficiently (for example, Rational Developer for System z, and Enterprise Generation Language).
- Second, smartphone users are notified when certain events occur on their mainframe computers. This model is not a new concept for example, e-mail notification), but we would like to demonstrate how we can implement a publish and subscribe model where information is pushed down to smartphone clients from a mainframe platform.

In the demonstrated scenarios we use a variety of tools, standards and technologies.

Attention: If you would just like to quickly experience a smartphone application accessing a z/OS back-end system, read Appendix A, “Test drive yourself” on page 165.

2

Architectural overview

In the previous chapter we introduced possible extensions to the mainframe platform in order to be up-to-date with new information technology developments. SOA and Web 2.0 are a powerful combination, and the usage of smartphones can improve the solution even more.

In this chapter we provide a brief overview of architecture viewpoints on a new landscape involving both the traditional mainframe environment and the new Web technologies introduced in Chapter 1, “Introduction” on page 1. These viewpoints are then incorporated in real scenarios that can be easily implemented.

The main architectural diagram (Figure 2-1 on page 10) is based on the classic logical three-tier architecture: a presentation tier, a business logic tier and a data tier. This logical three-tier architecture can be implemented as two physical tiers, where the middle tier is placed on System z, either in separate z/OS LPARs or in Linux on System z images.

There is also the possibility to have a logical two-tier architecture accessing back-end data or transactions directly. This could be a good option for many intranet applications, but not for applications that are used on the Internet, because of security.

Attention: If you have too little time to read on, you can jump to “Architecture quick guide” on page 15, where you can find the most important architectural choices regarding smartphone access to z/OS!

© Copyright IBM Corp. 2010. All rights reserved.

2.1 Architecture overview of our scenarios

Figure 2-1 shows the logical three-tier architecture that applies to many real world applications. We are also using this as a guideline in the various scenarios in this book.

The diagram not only shows the different logical tiers, but it also shows software components that are used in each tier. On the client side we assume that the client device is always a smartphone.

Important: Be aware that the smartphone capabilities are determined by the manufacturer, the operating system that runs on it, the communication capabilities, and the type of contract you have for the phone. So, just the fact that you have a smartphone may not necessarily mean that you can execute any or all of the scenarios discussed.

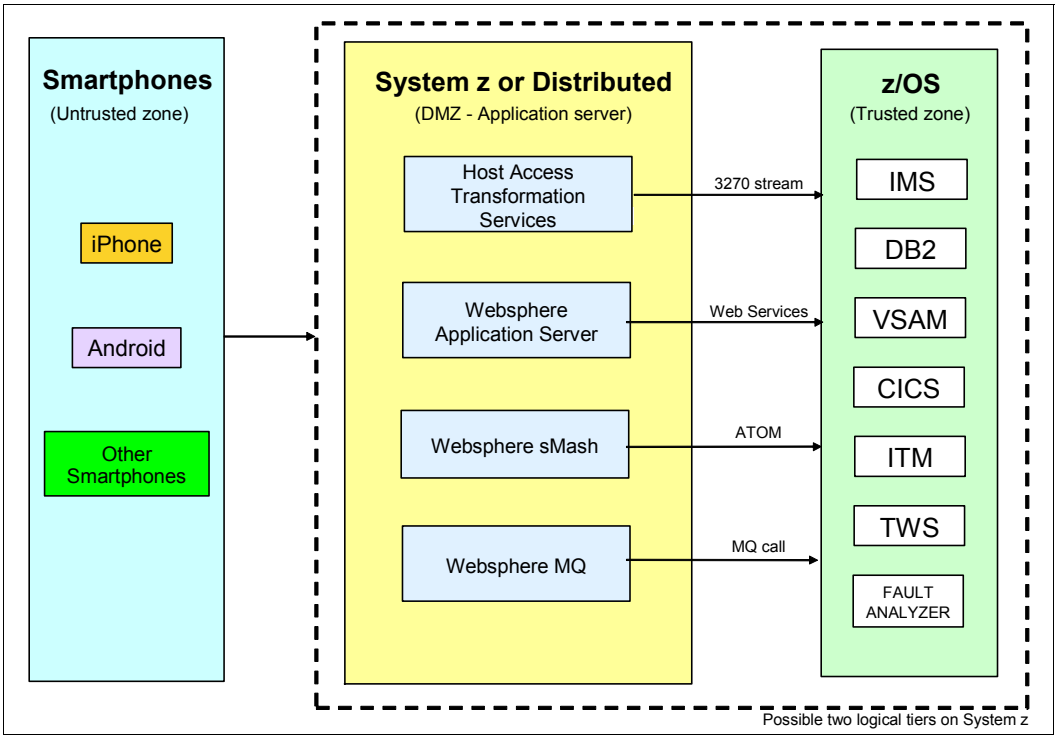


Figure 2-1 Architectural overview for extending z/OS applications to smartphones

Important: The components shown in the above architecture diagram are what we focus on throughout this book. There may be other components that we do not mention, but that could also play a role in this architecture.

The logical three-tier architecture described in Figure 2-1 is composed of the following main components:

- Smartphone component

  This is the presentation tier. It displays information on smartphone devices, and communicates with other tiers by sending requests and obtaining data. We considered several possible devices as presentation devices, and most of the scenarios we used can be easily adapted to different types of additional smartphones.

- Application server component

  This tier is typically used for controlling the application flow, composing the user interface logic and accessing the back-end tier. This tier can also be used to implement business logic. This layer is usually protected by firewalls and may be part of the *demilitarized zone (DMZ)*. We considered several options as possible application servers, from the simplest one (WebSphere sMash, which is easy to implement and based on open source) to a production one (WebSphere Application Server, which is stable, scalable, and ready for managing heavy workloads).

- Back-end systems component

  This is the business logic and data tier, where core business logic is executed and information is stored and retrieved. In our scenarios, in addition to traditional mainframe data subsystems we considered the possibility of accessing some infrastructure products, such as Tivoli Workload Scheduler (TWS), IBM Tivoli Monitoring (ITM), or Fault Analyzer.

As mentioned before, a logical three-tier architecture can be implemented on two physical tiers. We represented this configuration with the dotted line box around the middle and data tiers in Figure 2-1 on page 10. Application servers can be run on System z, either in a z/OS environment or on a Linux on System z image.

2.2 Web application or client application

In the following chapters we describe several possible scenarios in order to demonstrate real implementations of the general architecture we introduced in 2.1, “Architecture overview of our scenarios” on page 10.

The first selection to be made is related to the application type. You must determine whether you want to reach back-end systems by using a Web application or running a client application on the mobile device. Both these approaches have advantages and disadvantages.

For example, Web applications, running in the smartphone’s browser, are lightweight and fully integrated with Web 2.0, but native smartphone applications can give a much richer user experience compared to Web-based interfaces. We take a closer look at each in the next two sections.

2.2.1 Web application

Web 2.0 is based on the Web application paradigm. Web applications are lightweight, use agile protocols and can have the look-and-feel of desktop applications, using AJAX technology. Web applications fit perfectly in the standard logical three-tier layout for e-business applications. Because they use an application server on the host for running any business logic, security can easily be enforced by implementing firewalls and using a DMZ concept.

The Web application server infrastructure can be designed in two ways:

- By using a lightweight version of a Web application server (WebSphere sMash)
- By using a full-blown production-ready Web application server, such as WebSphere Application Server

The advantages of WebSphere sMash derive from the quick time to set up the infrastructure to satisfy a user request and the openness with regards to popular programming models such

as PHP, Groovy, Dojo, and so on. In this way, even a beginner can easily set up a Web application. There is also a repository with many free application demos. It is also possible to migrate these applications at a later time to the WebSphere Application Server or consolidate them to WebSphere sMash on Linux for System z.

Advantages of the WebSphere Application Server are stability, scalability, and all the characteristics required by a production environment. It is possible to use the infrastructure that is already in place to set up a Web application solution for smartphones. Many application architectures are already designed with a “multi-channel” concept, thus allowing mobile devices to access common application logic in the application server.

2.2.2 Client application

Client applications are another solution to accessing data from smartphones. A client application is an application running natively on the smartphone and that can be developed using smartphone-specific APIs. In general, this type of application can have more intelligence than a Web application. Using client applications means having some local logic on the device, so it is easier to manipulate data for local requests. This type of application can better use the native capability of the device’s operating system.

It is possible to use client applications in two ways: sending the requests from the client to the server (asking for data) or waiting in stand-by for a server to wake up, based on an event. In the following chapters we propose scenarios to implement both.

2.3 Communication styles

There are several communication styles and techniques that can be used to access back-end systems from smartphones. The choice should be based on the available products, the nature of the application, security requirements, and the available mainframe services you want to access.

This section briefly describes the access methods we used in our scenarios.

2.3.1 3270 data stream

The 3270 data stream operations are designed primarily for transmitting data between an application program and a 3270 terminal emulator with a keyboard so clients can interact with mainframe-based applications. It is the traditional way to access data.

We will use Host Access Transformation Services (HATS) to access mainframe data via 3270 data streams using existing z/OS applications.

2.3.2 Web Services

Web Services has become a common term in the IT industry. Web Services can be used to implement an architecture according to service oriented architecture (SOA) concepts.

Web Services are a set of emerging standards that enable interoperable integration between heterogeneous IT processes and systems. A *Web service* is a new breed of Web application that is self-contained and self-describing, and that can provide functionality and interoperability ranging from the basic to the most complicated business and scientific processes.[1]

2.3.3 Asynchronous messaging

Messaging queues offer solutions to connecting and integrating many applications including mainframe resources and mobile applications without applications being aware of who is the client.

In the scenario we demonstrate in this book, we used the MQ Telemetry Transport (MQTT)[2] publish/subscribe messaging protocol and the Really Small Message Broker, which is a very small messaging server that uses the lightweight MQTT publish/subscribe protocol to distribute messages between applications.

2.3.4 Atom feeds

Atom feeds are a new way to access data on the mainframe. Atom allows someone to link to not only a page, but to subscribe to the page with notification every time that page changes.

Atom is both a protocol and an XML format for content providers to provide XML-based Web feeds of updated content. An Atom feed is a Web feed that uses the Atom protocol and format. This provision of updated content is known as *syndicating* a Web feed. Web users can subscribe to a feed, allowing them to see new content as soon as it is made available.

Several IBM subsystems support Atom feeds. CICS was one of the first, and it has been supporting since CICS Version 3.2 with a support pack.

2.4 A different approach: logical two-tier architecture

In the previous sections we always assumed three logical tiers, but there is also a possibility to directly access the “back-end” tier from a smartphone. When doing this it is very important, however, to consider the security implications. At an extreme, a smartphone may be fully untrusted and the back-end data tier that you may want to access may be fully in the trusted zone of your IT environment. In that case, direct connectivity between the smartphone and the back-end on System z may not be recommended under any condition.

2.4.1 Logical two-tier with CICS

One of the back-end environments that come to mind first is CICS. CICS programs have had the ability to respond to HTTP requests with a variety of information for quite some time. In the case of providing HTML, Cascading Style Sheets (CSS), graphics, JavaScript, and so on, CICS' support is, in concept, the same as providing them from the WebSphere Application Server. The difference is that CICS also supports multiple programming languages, including COBOL, PL/I, C, C++, Java, Assembler, and PHP.

When composing any CICS-based application, you will always want to separate the presentation logic from the business or data logic (IBM has been recommending this to CICS customers for over 15 years). Some people look at these as different tiers. Dividing the application this way is the same concept as using a servlet for presentation and an EJB, CICS, SAP, or some other back-end system for business logic and data access. CICS provides support for both inbound and outbound Web Services, so if the presentation logic needs to invoke a Web service somewhere (.Net, WebSphere Application Server, anywhere),

[1] Excerpt from http://www.ibm.com/developerworks/webservices/standards/
[2] See http://mqtt.org for more information

you could do that also. The Web pages CICS can send out can include as much JavaScript, widgets, mashups, etc as you would like. Some people consider that a different tier.

CICS Transaction Server v4.1 also supports the Service Component Architecture (SCA), so the ”front-end” presentation logic can just do an INVOKE SERVICE, and not care whether that ”service” is a CICS program running on the same CICS region, different region, or is really a Web service request to a Microsoft® environment. It does not need to care. As with any SCA environment, the resolution of the service is done in a wiring diagram before the composite is deployed. CICS uses a new *bundle* resource for this, similar in concept to JEE packaging, but is CICS-specific. Multiple tiers can reside on the same hardware and even in the same software infrastructure. Often, tiers are logical things, not physical things.

CICS applications can also provide RESTful interfaces, Atom feeds, and Web services. CICS can also send any kind of requested file (CSS, JavaScript, graphics, applets, whatever), and you can upload files via the Web to CICS also. HTML sent out from CICS can contain JavaScript, do AJAX, include widgets, be mashups, and so on. CICS supports Web 2.0 and also has support for eight programming languages, but unfortunately many times people remember the way it worked 40 years ago and look no further. CICS has built-in support to return FILE and TSQueue data as an Atom feed, and also provides assistance for exposing DB2 or other information as a feed.

CICS is fully compliant with HTTP 1.0 and 1.1 in the role of the HTTP client and HTTP server. CICS is not an HTTP tunnel, proxy, caching server, or any of the other roles described in the HTTP 1.1 specification. The client and server roles are *must* levels, the other roles are *should* levels. When you do all the musts, but do not do all the shoulds, the technical term applied is ”conditionally compliant”. CICS can also do SSL and TLS up to 256-bit encryption, can do client certificate exchanges in both the client and server role, and also supports HTTP Basic Authentication (again both inbound and outbound).

To access a very simple application, you can use the following:

```
http://zserveros.demos.ibm.com:8082/account
```

This sample application uses CICS TS V3 to provide a user interface to create, read, update, delete, and browse account information stored in a VSAM file. CICS provides the graphics, JavaScript, Cascading Style Sheets, and HTML for the Web pages that make up this Web browser user interface.

The application (which happens to be written in COBOL) looks at the HTTP headers sent from the Web browser. If the User-Agent header (which indicates the type of Web browser that made the request) contains iPod or iPhone, then it will send out HTML that references a Cascading Style Sheet for the very small iPhone screen (which causes the menu to be a single line across the top instead of down the left side). Additionally, this application sends a meta tag that tells the iPhone to use the iPhone size resolution instead of trying to pretend that it is a ”big browser” (which it does by default).

2.5 A word on security

It is a safe assumption to say that, by default, smartphones are unsafe or at least untrusted, unless the smartphone has been set up with significant security features installed and security options enabled. On the other hand, a smartphone would connect to your application infrastructure, just like any other computer. So, any security design you would have set up already to allow trusted and untrusted computers to access your infrastructure would also apply to smartphones. The difference lies in the fact that a smartphone is more vulnerable to loss or theft than a regular computer located in a home or office.

Security technology support on the various smartphones models and their operating systems varies. Things are not as simple as with regular computers, which in most cases are running Microsoft Internet Explorer browser. In the best case, however, you may be able to get your smartphone to work with SSL, Basic Authentication, and even VPN. So, you may be able to implement sufficient security to let a smartphone safely connect to a real business application on a server. But again, you have to give a lot of thought to what could happen if the phone gets lost and somebody gets hold of it. You should not have any passwords cached, for example, and a (VPN) connection should be disabled automatically after 5 or 10 minutes of inactivity.

2.5.1 Security in our architecture

When using a three-tier architecture, the most important action is to secure the communication between the smartphone and the middle tier. The middle tier will typically be a “WebSphere” type of middleware with extensive support for secure connections, supporting SSL and various authentication methods at a minimum. In a more robust design the middle tier will probably consist of multiple layers and a Secure Proxy Server, as well as a number of routers.

Attention: In the remainder of this book we do not focus on security. This is not because we did not find it important, but because we were not able to contain it in the scope. All best practices, considerations and concerns with regards to security apply as they would apply to any other Internet or intranet security design. When implementing smartphone access to your mainframe environment, you should consult other publications covering security in Web architectures.

Security in a two-tier architecture

When there is no middle tier, but only a CICS, IMS or DB2 back-end on z/OS, there are other considerations. If the application is an intranet application and an employee connects to it over a VPN, you should focus on the enabled security features on the smartphone. If the application or database contains sensitive data, you may not want to allow smartphone access at all. If the application is accessed over the Internet, for example, a user checking in on a flight or changing his/her reservation and accessing the mainframe application directly, focus should be on the authentication method used and the usage of SSL. Again, certain back-end applications on z/OS may not be suitable for smartphone access at all and it may not be possible to guarantee sufficient security from a smartphone client perspective.

2.6 Architecture quick guide

After your company’s business unit has made the decision that offering certain functionality on a smartphone would benefit their business, it is time to assess the impact of such a solution on your application architecture and infrastructure.

Important things to know are:

- Is there already a Web infrastructure in place? And, if so, is this infrastructure multi-channel enabled?
- What level of security does the new application need to have?
  Many organizations work with a certain categorization of security required by the application, from low to very high.
- What development skills are available and what development standards are to be followed?

- What communication style does the new smartphone application require? Is the communication always triggered or started by the user, or is there a requirement for a “push” scenario, based on publish/subscribe or event processing?
- Does the user interface need a lot of built-in intelligence or is there an expectation that there will be a lot of data pulled into the smartphone?
- What are the technological capabilities of the back-end application on z/OS? These capabilities are typically determined by the middleware in which the application runs.

2.6.1 Decisions to make

Based on the requirements discussed before, you may be limited to certain implementation options.

Decisions you need to make with respect to any scenario of a smartphone accessing a z/OS environment are:

- Do you need a native client application on the smartphone or is a Web application sufficient?

  The following considerations play a role in making a choice:

  - Maintenance
    A Web application is easier to maintain because the code is kept on the server. If there is a new version of the application it will simply be downloaded as soon as it is upgraded on the server. A native application would need a mechanism to refresh the application on all of the users that installed it.
  - Functionality
    A native application offers more functionality and more built-in intelligence to the client. Depending on the smartphone operating system, complete API sets are available to do the most sophisticated things you can imagine.
  - Security
    A Web application benefits automatically from browser functionality, especially security features, such as SSL.
  - Resource consumption
    Resource consumption on a smartphone is still an important consideration, as smartphones have limited memory on board. A native application will use some memory on the smartphone, even when not used. A Web application will only use some memory when actually used, and then this memory will probably be less than with a native application.

- How will the smartphone communicate with your server?

  Note: The assumption is that for any of the scenarios discussed in this book or similar ones, the smartphone is enabled for data communication. Data communication can take place in two ways:

  - Using wireless connectivity ("wi-fi"). To be able to use wi-fi, the smartphone must have wi-fi support. Be aware that not all smartphones have wi-fi capabilities. With wi-fi support your smartphone is capable of connecting to a wi-fi network through a "hotspot". Once connected to the wireless network you can access the Internet through the wireless network service provider.
  - Using the cellular network provider's network and a so-called "data plan". Practically all cellular network providers offer data plans. A data plan can be limited, for example, 20 MB per month, or it can be unlimited. Be aware that the speed of the network can vary enormously, depending on where you are. Using a data plan allows you to surf the Internet anywhere you are, independent of availability of wireless ("hotspot") networks.

  – Physical infrastructure

    The smartphone client application will need to access your infrastructure at some point. How this happens and where depends on how your existing Web application infrastructure is set up and whether there are additional requirements. For example, if you already have an infrastructure serving browsers on PCs and you have routers, proxy servers, firewalls, DNS, HTTP servers, Web application servers, and so on, already in place, the most logical starting point is to let the smartphone communication come into this same infrastructure.

  – Application architecture

    From an application architecture perspective you need to decide on the communication protocol and style. Examples are discussed in this book. The most common ways of communication today are HTTP, Web Services, and Web 2.0. Note that Web Services is only an option when using a native client. HTTP requires an HTTP server on the server side and Web 2.0 requires Web 2.0-enabled middleware on the server.

    Regarding communication style, you have to decide whether you need synchronous or asynchronous communication. Browsers are by definition asynchronous. Also, you may have requirements where the smartphone is "sleeping" and wakes up when the server "pushes" down events, alerts or messages. These messages can be e-mails, text messages, or even MQ type messages. A native application has the advantage that it can keep on running in the background on the smartphone while waiting for messages to come in.

- And what about security?

  We already had a discussion about security in "A word on security" on page 14, and it is very likely that security requirements are the most determining and limiting factor in choosing your architecture.

  Attention: We are not able to make any statement about what works and what does not work on smartphones in general or any smartphone in particular. This depends on the support and features provided by the smartphone manufacturer, its operating system and, in case of Web applications, the browser. Especially with respect to security technologies, it also depends on how compatible the middleware on the server and the smartphone is.

Security requirements could be:

- Authentication
  Weak, strong, or somewhere in the middle. This will depend on the application that will be accessed on the z/OS server. Basic authentication is easy to implement on most smartphones, especially when using a full-function browser on the smartphone and a mainstream HTTP server or Web application server product on the server. Anything beyond that can become more difficult and fully depends on the specific functions and features of the smartphone and its software.
- Encryption
  Being able to use SSL can very quickly become a key requirement. As soon as sensitive data is being transferred between the server and the smartphone, SSL is required. Again, when using a full-function browser on the smartphone and a mainstream HTTP server or Web application server product on the server, this should not be difficult to enable.
- VPN
  Especially for corporate intranet application access, VPN can become a requirement. Using VPN on a smartphone is possible, and some companies have already enabled their mobile employees to access the corporate intranet using a smartphone. However, VPN solutions on smartphones are very specific and only certain VPN software may work well on certain smartphones. Once you are connected over VPN, you can surf the intranet from your smartphone. You may also be able then to access core business applications on z/OS using a terminal emulator, or even directly using any of the techniques discussed in this book (for example, Chapter 7, “Accessing a CICS Restful Web service from Apple iPhone” on page 125).

- How to develop the smartphone application?

  There are all kinds of tools and workbenches available to develop applications for smartphones, but the most important thing to realize is that native smartphone client applications are client-specific. An application developed for an Apple iPhone does not run on a device with Windows Mobile. So, if you want to provide a native smartphone application and you want all your potential users to be able to use it, you may have to provide 5 or 6 versions of it.

  This problem is significantly less serious when developing Web applications, running inside the smartphone’s browser, even though there is no guarantee that a Web application will work seamlessly in all mainstream browsers on the smartphone market. Some Web applications are “optimized” for a certain browser, for example Safari, and may not work properly in other browsers. It is possible to build some intelligence into the Web application, so that it can recognize which browser the user is using and then continue sending down optimized pages for that specific browser.

  For Apple iPhone and Android there are special toolkits available to develop native applications. Some Web development workbenches support development for smartphones and, as discussed in Chapter 6, “Accessing a CICS Web service with an EGL Rich UI client on Apple iPhone” on page 93, Enterprise Generation Language (EGL) can be used too.

- And what to do on the server side, on z/OS?

  The most important point is that the back-end application on the z/OS server supports the communication protocol and style used by the smartphone while adhering to all the security requirements. This will depend on the middleware on which the application runs.

  The most common architecture, however, will be an architecture where the smartphone communicates with a “middle-tier”, typically reaching an application server or ESB, after having gone through layers of routers, firewalls, and proxy servers. This middle-tier then again communicates with a CICS or IMS backend, or directly accesses a database.

The middleware component that directly communicates with the smartphone needs to be able to work with communication standards used by the smartphone.

2.7 The scenarios in this book

In this book we demonstrate the following scenarios:

- In Chapter 3, “Accessing z/OS applications with HATS from Apple iPhone” on page 21 we show how you can use Host Access Transformation Services (HATS) to access a back-end system through a 3270 data stream and render a smartphone interface.
- Chapter 4, “Accessing CICS in Atom feeds format using WebSphere sMash” on page 51 shows how you can use the WebSphere sMash platform to perform rapid application development for accessing a CICS back-end system on z/OS.
- In Chapter 5, “Accessing mainframe resources from Android mobile device applications” on page 73 we focus smartphone client applications with deeper functionality such as the ability to receive messages from the z/OS host system. The samples in this chapter are all based on the Google Android platform.
- Chapter 6, “Accessing a CICS Web service with an EGL Rich UI client on Apple iPhone” on page 93 demonstrates the power of Enterprise Generation Language (EGL) Rich UI in developing user interfaces for the smartphone.
- In Chapter 7, “Accessing a CICS Restful Web service from Apple iPhone” on page 125 we focus on a scenario in which we access a CICS Web service from a smartphone using REST.
- In Chapter 8, “Accessing IBM Tivoli Monitoring from smartphones” on page 151 we show how you can use a smartphone application to look at the status of your z/OS system from a systems management perspective.

2.8 Self-service examples

You can try some smartphone applications accessing a z/OS system directly without installing anything or even registering for anything! Refer to Appendix A, “Test drive yourself” on page 165.

3

Accessing z/OS applications with HATS from Apple iPhone

You can extend all or selected functions of 3270-based applications to mobile phones, data collection terminals and personal digital assistants (PDAs) with IBM Rational Host Access Transformation Services (HATS).

In this chapter we discuss extending z/OS applications to smartphones with the help of HATS. We take you through step-by-step instructions to enable your application as a HATS Web application and customize it for smartphone devices.

Important: Note that Android Smartphone devices are currently not supported by IBM Host Access Transformation Services (HATS).

iPhone is supported as of IBM Rational Host Access Transformation Services for Multiplatforms and 5250 Applications V7.5.1. You can find the announcement letter here:

http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&appname=iSource&supplier=897&letternum=ENUS210-140#

© Copyright IBM Corp. 2010. All rights reserved.

3.1 High-level architecture

In this scenario we connect to a mainframe system directly from our smartphone using realtime synchronous access. The basic solution architecture is shown in Figure 3-1.

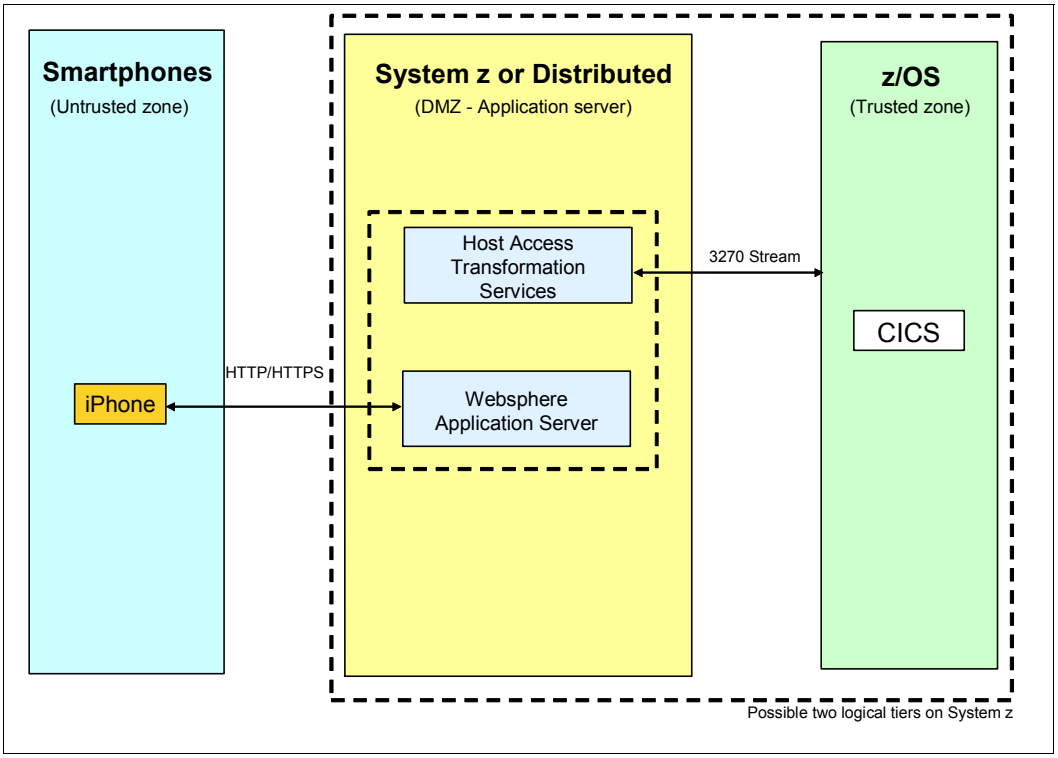


Figure 3-1 Solution architecture: Accessing z/OS applications from smartphone devices using HATS

The process for developing a HATS Web application for a mobile device is the same as developing any HATS Web application, but requires consideration of the following characteristics:

- Smaller screen size
- No mouse
- No or limited keyboard
- Limited processing power

3.2 Software used

We used the following software products to develop, deploy and run this scenario. We discuss each of these products briefly in the coming sections.

- Host Access Transformation Services Version 7.5
- Rational Developer for System z Version 7.6
- WebSphere Application Server Version 7.0
- CICS Transaction Server Version 3.2

3.2.1 Host Access Transformation Services (HATS)

IBM Rational Host Access Transformation Services (HATS) offers tools to help you build integrated Web applications from existing terminal applications, thereby making your users more productive and less error-prone. Because HATS creates a standard Web application, the only client software necessary for accessing the application is a standard Web browser. Access through a Web browser allows extension of all or selected functions of the green-screen applications to intranet, extranet, and Internet users.

The latest releases of HATS also support mobile device clients running Internet Explorer Mobile. So, after modernizing your existing applications with HATS, you can extend your existing z/OS applications to mobile devices or smartphones, thus providing users with real-time information that will alleviate errors and increase customer satisfaction.

HATS consists of two components: the HATS Toolkit and the HATS runtime. We discuss these in the following sections.

HATS Toolkit

The HATS Toolkit provides a set of wizards and editors that guide you through creating and modifying HATS applications. The toolkit is integrated with the Eclipse-based IBM Rational Software Delivery Platform. This platform—which includes Java Platform, Enterprise Edition (JEE), Java, team, Web (including Web page design), and portal development capability and multiple integrated test servers—enables you to develop and test a full-function application that matches your corporate look and feel in one development environment before deploying to the server. For deployment directly to IBM WebSphere Application Server, you would package the HATS application as a JEE Enterprise archive (EAR) file. For deployment to IBM WebSphere Portal Server, you would package the HATS application as a Web archive (WAR) file.

HATS runtime

The HATS runtime is a set of servlets and classes automatically packaged with the HATS application. The HATS runtime:

- Manages connections to the Telnet servers for the terminal applications that you are transforming.
- Tracks users and their interactions with one or more HATS applications.
- Provides a transformation servlet (HATS entry servlet) that processes requests from users and dynamically transforms terminal data to HTML based on the request, the screen received from the terminal application, and the customization that the developer defines
- Provides an Administrative Console servlet that enables connection management and problem determination for one or more HATS applications.

For more information about HATS, visit the following Web site:

`http://www-01.ibm.com/software/awdtools/hats/`

3.2.2 Rational Developer for System z (RDz)

Rational Developer for System z (RDz) consists of a common workbench with an integrated set of tools that offers application development and maintenance, run-time testing, and rapid development and deployment of simple and complex applications. It offers an integrated development environment (IDE) with advanced, easy-to-use tools and features to support application development for multiple runtimes such as CICS, WebSphere, IMS, and DB2. It helps developers rapidly design, code, and deploy complex applications.

RDz provides support for Cobol, Assembler, PL/I, Java, JEE, C/C++, SQL and stored procedures.

The HATS Toolkit uses the basic set of tools offered by RDz for application development purposes, so the HATS Toolkit needs to be installed in the Eclipse-based Rational integrated development environment (IDE). For this scenario, we used RDz as a base for the HATS Toolkit. You can also install the HATS Toolkit with other Eclipse-based Rational IDEs such as Rational Application Developer or Rational Business Developer.

For more information about Rational Developer for System z, visit:

http://www-01.ibm.com/software/awdtools/rdz/

3.2.3 WebSphere Application Server

WebSphere Application Server is a runtime environment for Java-based applications. The application server acts as middleware between back-end systems and clients. It provides a programming model, an infrastructure framework, and a set of standards for a consistent integration between them.

WebSphere Application Server is available on a wide range of platforms. WebSphere Application Server V7.0 provides the runtime environment for applications that conform to the J2EE 1.2, 1.3, 1.4, and Java EE 5 (formerly J2EE 1.5) specifications.

Once we have developed applications using RDz and the HATS Toolkit, we use WebSphere Application Server for testing and deployment purposes. As discussed earlier, HATS applications can be exported as a JEE WAR (Web archive) or EAR (Enterprise archive) file from the HATS Toolkit and can be deployed to WebSphere Application Server for testing and production purposes. RDz can also connect to a remote WebSphere Application Server for application publishing or testing.

For more information on Websphere Application Server, visit:

http://www-01.ibm.com/software/webservers/appserv/was/

3.2.4 CICS Transaction Server

In this scenario we accessed a CICS region on z/OS. The CICS region needs to have the back-end application installed and running. In our case, we used a program called CUSTINQ, with CICS transaction code IINQ. In your case you may have another CICS program to use, in which case you would need to follow the steps with some adjustments.

3.3 Creating a HATS application for Apple iPhone

In this section we discuss the steps necessary to create a HATS application for iPhone.

Important: As described in this chapter, we could easily get a HATS application to work with an iPhone client using an earlier version of HATS. However, use of an iPhone client device with HATS is only supported by IBM as of HATS Version 7.5.1.

Important: To follow this scenario, you need to have the previously discussed software components installed.

3.3.1 Sample code

The code developed for this scenario is available for download in both EAR format and as a Project Interchange File. Refer to Appendix D, “Additional material” on page 203 for instructions.

3.3.2 Getting started

Follow these steps to get started:

1. Start Rational HATS Toolkit 7.5 from **Start** → **Programs** → **IBM Software Delivery Platform** → **IBM Rational HATS 7.5** → **HATS Toolkit 7.5**.

   It will ask you to select a workspace. A *workspace* is a folder in your file system where you will store all the artifacts related to your project. Specify location to any folder in your file system, as shown in Figure 3-2.

   Click **OK**.

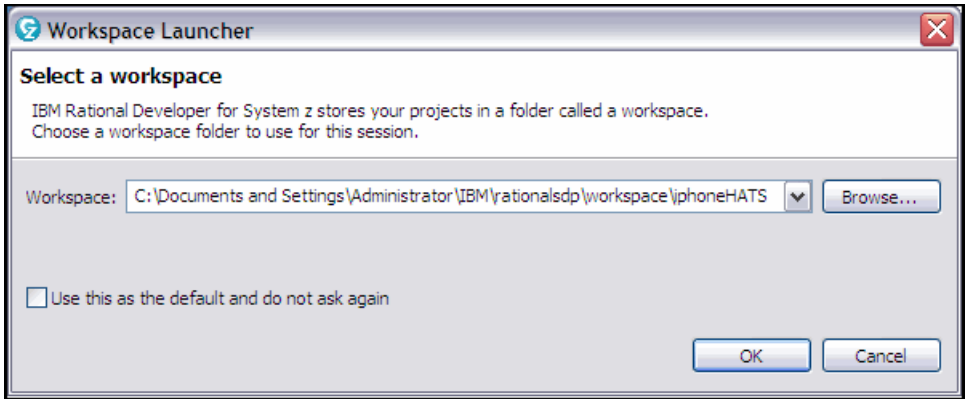


Figure 3-2 Selecting the workspace in the HATS Toolkit

   The workbench will be launched and you will see the HATS Toolkit open with the Welcome panel.

2. Now, switch to the **Host Access Transformation Services** perspective. You can do so by selecting **Window** → **Open Perspective** → **Other...** from the menu bar of your recently opened workbench. You will see a pop-up window on the panel for selection of the perspective as shown in Figure 3-3 on page 26.

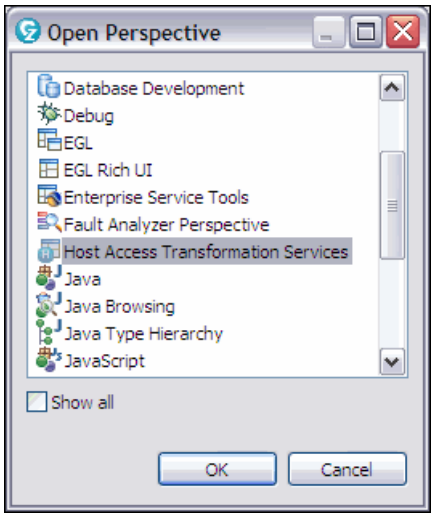


Figure 3-3 Selection of perspective (pop-up window)

3. Select **Host Access Transformation Services** from the pop-up window and click **OK**. You will see a change in layout as you just switched to the HATS perspective.

   You may see the Welcome to HATS! (HATS Tip) pop-up window. Just click **OK** to close it.

4. To create a new HATS project, select **File** → **New** → **HATS Project** (Figure 3-4).

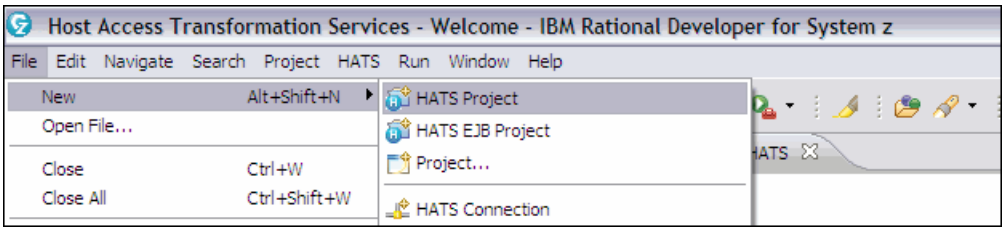


Figure 3-4 Creating a new project

5. In the pop-up window you need to specify the following information, as shown in Figure 3-5 on page 28:

   – **Name:** HATSiPhoneApp

   – **Description (Optional):** This application will provide access to the z/OS application on iPhone

   – **Deployment:** Web

   – **Target Server:** Choose WebSphere Application Server 7.0 from the drop-down, as you will use WebSphere Application Server 7.0 for testing and deployment purposes.

   – **Enterprise Application Project:** CUSTADDIPHONE_EAR

   – Select the check box for **Optimize options for mobile phone**.

     By selecting “Optimize options for mobile devices” in the Create a Project wizard, you can automatically limit the available HATS templates to only those optimized for mobile devices and set defaults to those appropriate for use with mobile devices.

     Projects created for mobile devices differ from typical Web applications in several ways. For example, in the project settings, a second rendering set named “compact” is created and set as the default.

In the compact rendering set, several defaults are used:

- To preserve space, the drop-down (selection) widget, instead of the link widget, is used for selection lists.
- Recognition and transformation of dialogs is activated and only the dialog area of the panel is transformed for display on the device.
- The application keypad, which provides application-level functions, is displayed as icons.
- The host keypad, which provides functions typically available from a host keyboard, such as function keys or the Enter key, is displayed as a drop-down list.
- To reduce the amount of HTML and blank space displayed on the PDA, the field widget uses the “separated” layout setting to render the output using in-line span tags to differentiate between fields.

In addition, to aid in displaying table data, HATS provides a columns placement function that allows the arrangement and exclusion of columns from the display, as well as expandable detail columns, to help you fit a table into a small space.

By default, only the first two columns are displayed as primary columns and all columns after the second are treated as detail columns. This enhancement allows tables to be collapsed and expanded, so that more data can fit without horizontal scrolling.

As an additional option, the table widget will create the expandable areas for retrieval by the browser only when requested. This approach uses AJAX technology and reduces the memory and footprint size on the client device. This feature can also be used in standard HATS Web applications.

With compact rendering, the main entry fields are reduced to fit on a smaller panel, yet are large enough to be easily usable without scrolling.

– Select the check box **Add administrative console support**.

The *HATS Administrative Console* can be used to manage connections and perform problem determination. You can include the HATS Administrative Console support files with the set of HATS applications contained within an Enterprise Archive (.ear) file deployed to WebSphere Application Server. In Figure 3-5 on page 28, you can see that the EAR file is called CUSTADDIPHONE_EAR. The deployed EAR file provides a runtime environment, in which the HATS Administrative Console enables you to manage HATS applications. If you include the HATS Administrative Console files in multiple projects, more than one administrative console can be started using different application names; however, each administrative console provides the same views and functions.

Instead of including the HATS Administrative Console support files in each EAR file, you can also create an EAR file that contains only HATS Administrative Console files. When you deploy this EAR file, you can start the HATS Administrative Console and change the management scope to enable HATS applications from multiple EAR files with one instance of the administrative console.

The deployed HATS Administrative Console project gives you a single point of administration for all HATS applications. When you create a new HATS project, you specify whether to include HATS Administrative Console files in the project. If you create a HATS Administrative Console project, you no longer need to add the administration support files to other projects you create.

The HATS Administrative Console is bound to WebSphere security. This means that when the WebSphere Application Server security is enabled, your system administrators must have proper authentication to perform Host Access Transformation

Services administrative tasks. The security functions used by the Host Access Transformation Services in WebSphere are based on the J2EE form-based authentication process when the WebSphere Application Server security is enabled.

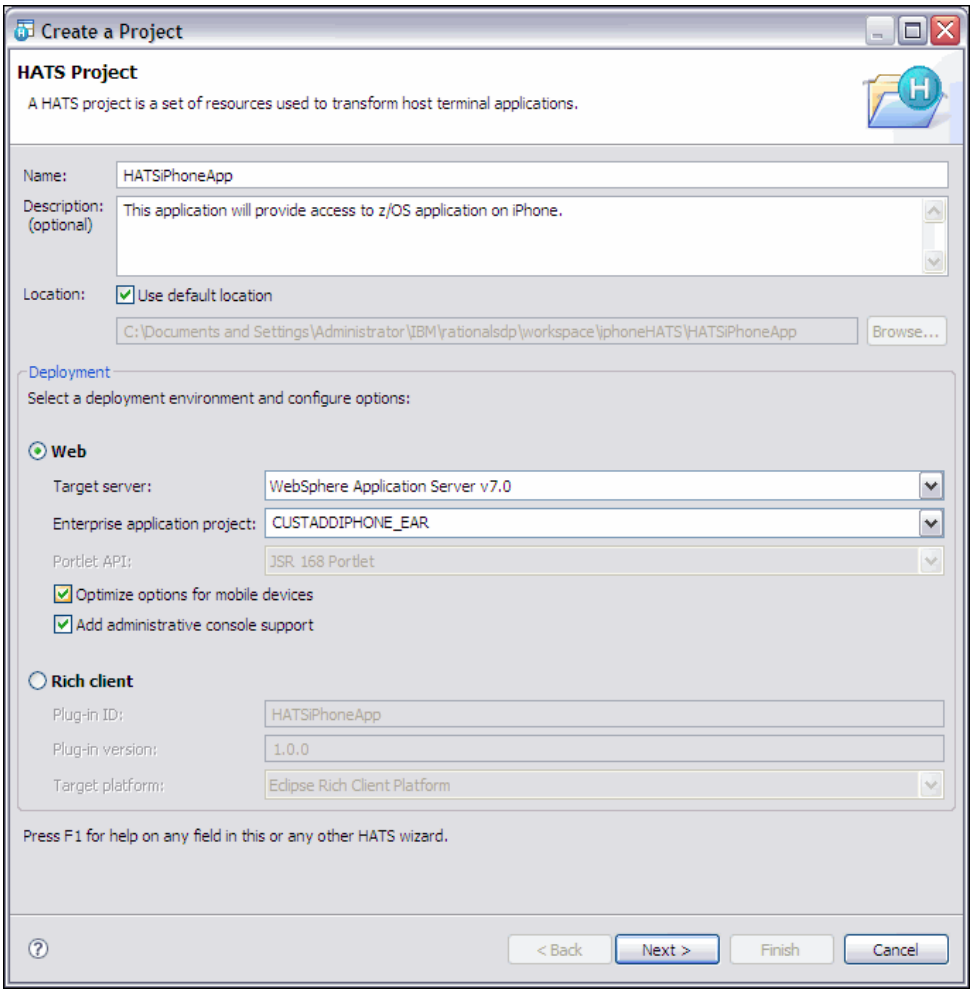


Figure 3-5 Creating the project: enter name, description and customize target type

6. Click **Next**.
7. You now have to enter host system connection information on the next panel, as shown in Figure 3-6 on page 29. In your situation use values that apply to your host environment. In our example we used the following values:

   Host Name: wtscz1.itso.ibm.com

   Type: 3270 (we are working on a z/OS 3270 data stream)

   Port: 23

   Code page: 037 United States

   Screen size: 24x80

   Again, these settings may be different for your host system; check with your system administrator for the correct values.

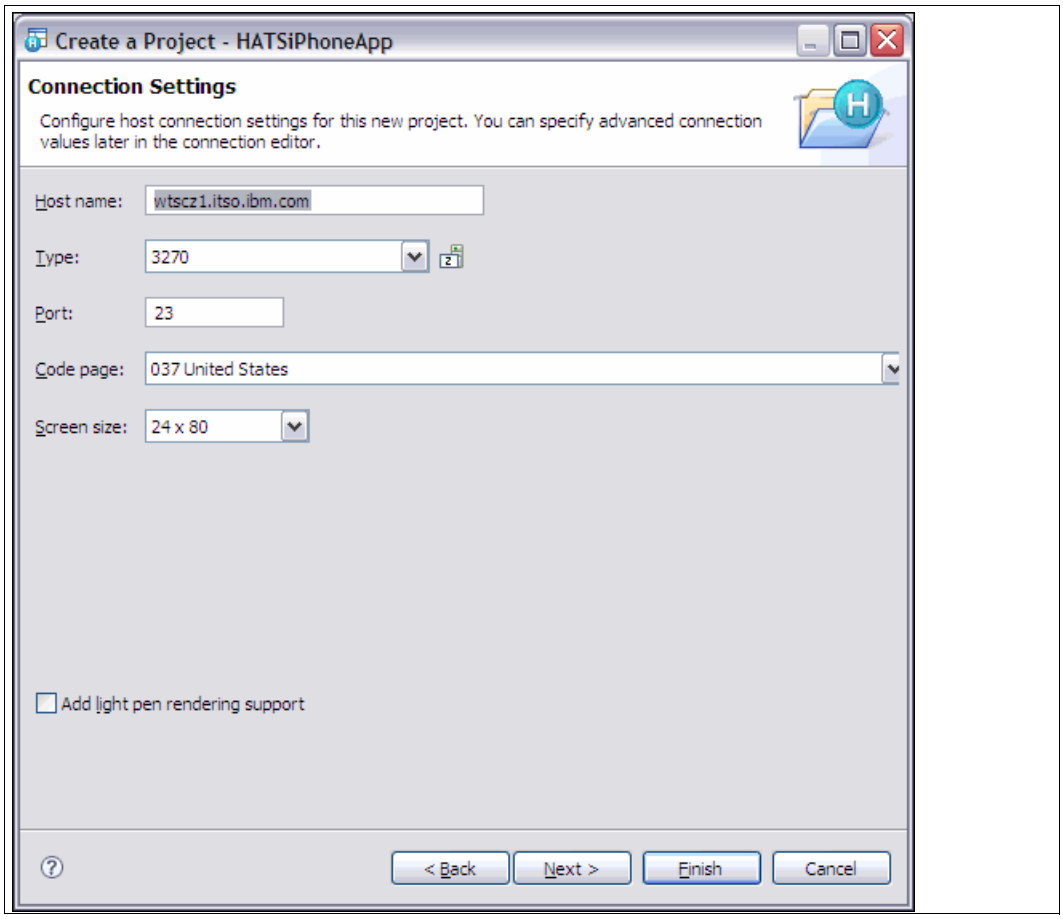


Figure 3-6 HATS Project Connection Settings

8. Click **Next >.**
9. You can now choose a specific template from the Template list. In our example we selected the Black template for our project, as shown in Figure 3-7.

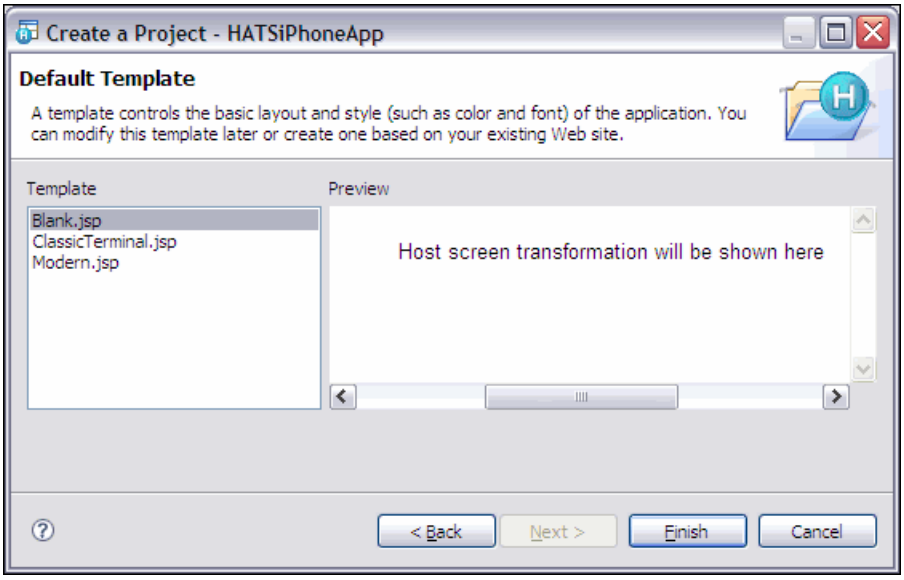


Figure 3-7 HATS Project Template selection

10. Click **Finish.**

    You may see a HATS Tip window with `Congratulations! You've created a HATS Project`**.** You can click **OK** to close it.

11. Once the HATS project is created, you can explore all the artifacts related to the project in the **HATS Projects** view, as shown in Figure 3-8.

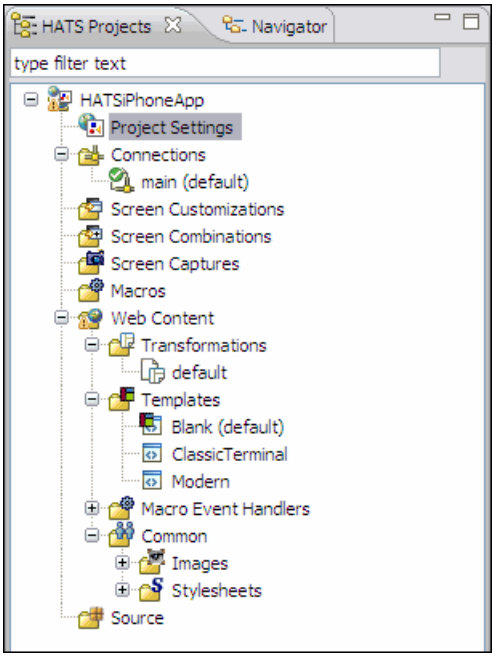


Figure 3-8 - HATS Project explorer view

12. Expand the **HATSiPhoneApp** project and double-click **Project Settings.** This will open up a new window with the **HATSiPhoneApp Settings** page, where you can modify any settings related to the project if required. See Figure 3-9 on page 31.

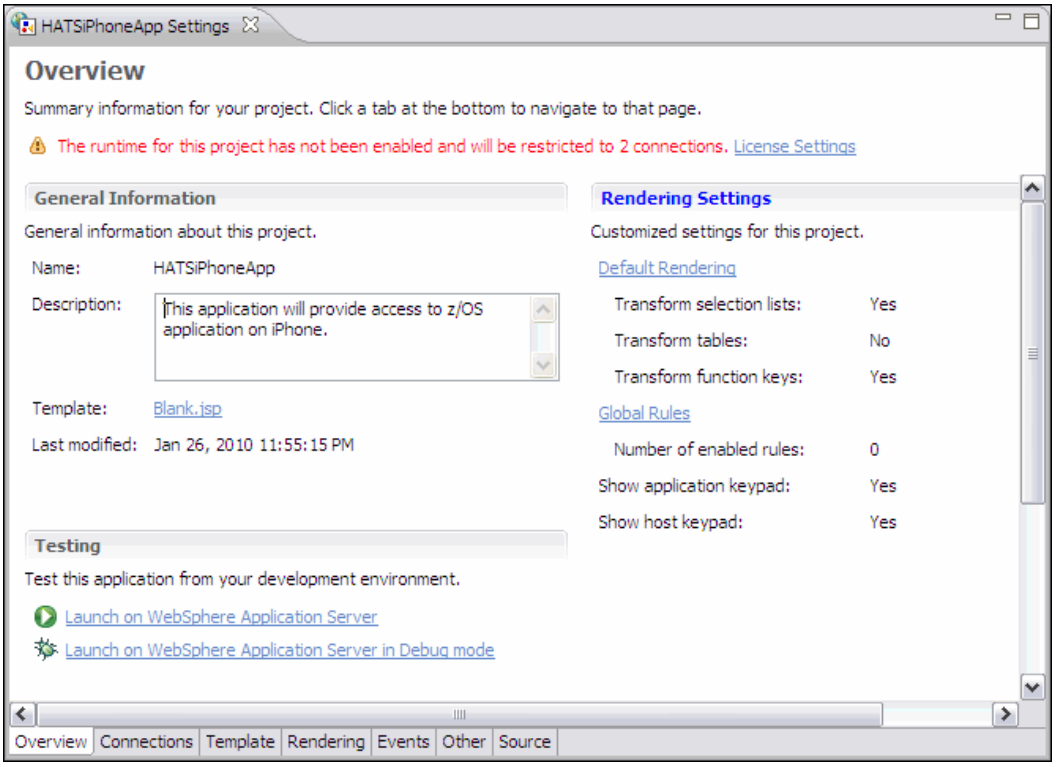


Figure 3-9 HATS Project settings page

13. Expand **Connections** from the HATS Project explorer and double-click **main (default)**, as shown in Figure 3-10.

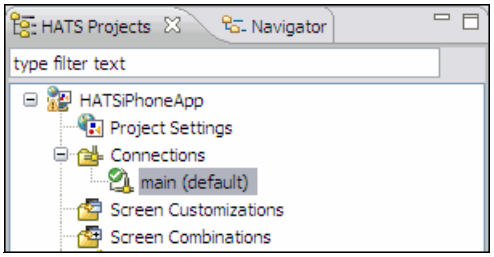


Figure 3-10 main (default) in the Project Explorer

This will open up a new page for connection settings. You can configure host system connection settings from here that include connection pooling, security, as well as user profiles. See Figure 3-11 on page 32.

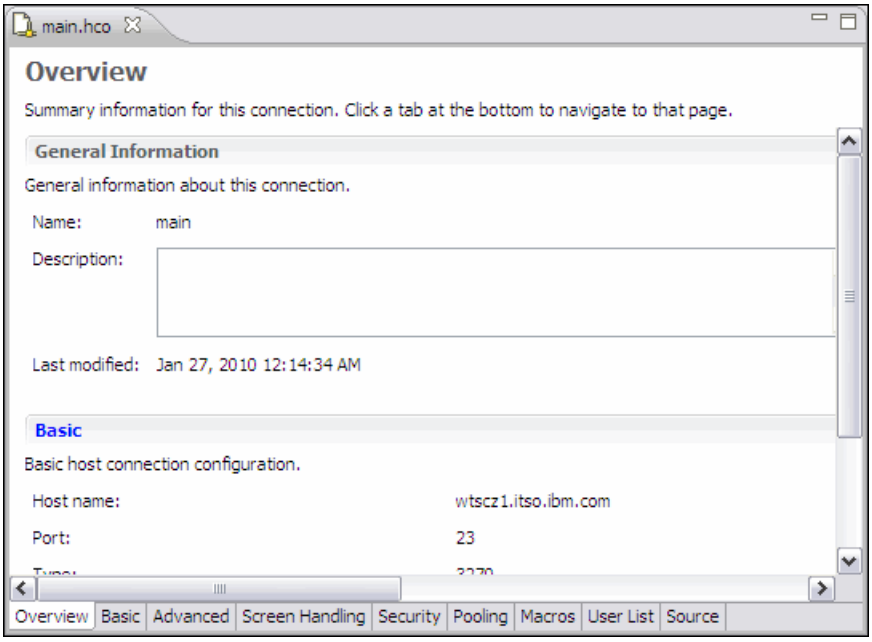


Figure 3-11 Configuring host system connection settings

You can right-click **main (default)** again, as shown in Figure 3-12, and select **Open HATS Host Terminal** → **main** from the pop-up context menu.

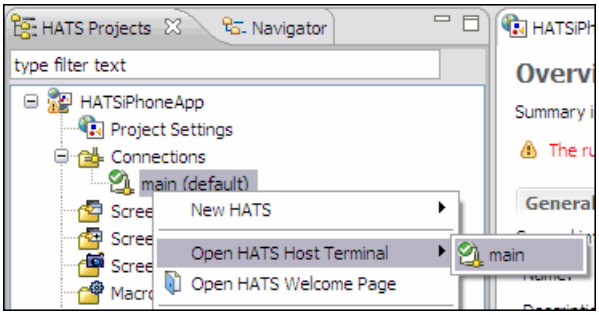


Figure 3-12 Opening the host terminal

This will launch a host terminal, as shown in Figure 3-13 on page 33. The host terminal toolbar can help you with the following tasks:

- Create a panel capture
- Create HATS panel customization
- Create HATS panel combination
- Record a macro

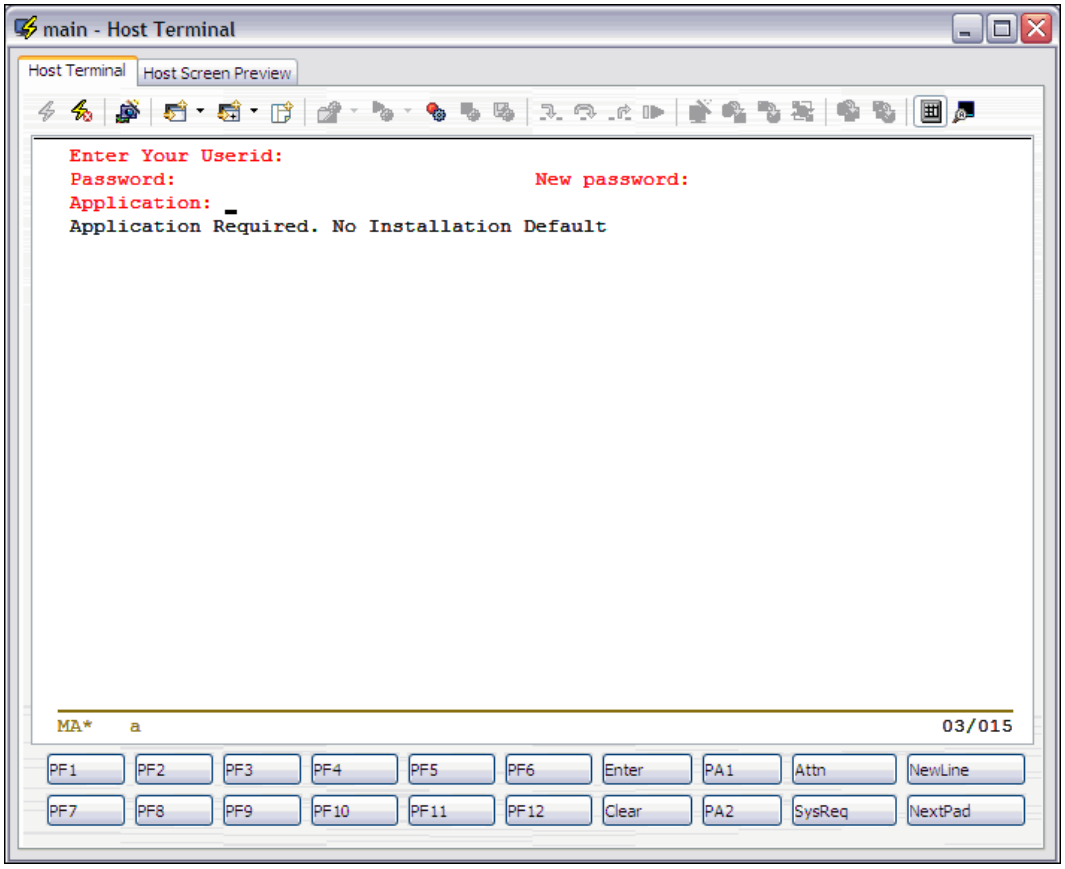


Figure 3-13 Host Terminal window

3.3.3 Customizing the template and CSS for an Apple iPhone

In this section we describe how to customize the theme of the Apple iPhone application to match the Apple iPhone Web standards. There are just a few changes that need to be made:

1. Expand **HATSiPhoneApp** → **Web Content** → **Templates** in the HATS Project Explorer and double-click **Blank (default)**.

   This opens up the Blank.jsp page in the JSP editor, which provides Design, Source, Split, and Preview tabs to work on JSP pages. Click the **Source** tab. You will see source code for the default.jsp page in the JSP editor.

2. Now add the meta tag, as shown in Example 3-1, in the HEAD section of the page.

   Example 3-1 Meta tag to add to JSP

```
<meta name="viewport"
content="width=320,user-scalable=no">
```

   The following JavaScript might be needed to display the HTML page as desired:

   – In case you want to remove the address bar from the browser and want the application to look like a native Apple iPhone application, you can add the following JavaScript in your HTML page (Example 3-2).

   Example 3-2 JavaScript to hide the address bar in the HTML page

```
<script type="application/x-javascript">
    if (navigator.userAgent.indexOf('iPhone') != -1)
```

```
    {
        addEventListener("load", function()
        {
        setTimeout(hideURLbar, 0);
        }, false);
    }
    function hideURLbar()
    {
        window.scrollTo(0, 1);
    }
</script>
```

– If you want to add a custom icon for your application when a user bookmarks the application to the home panel, you can do so by adding the following JavaScript in the HTML page (Example 3-3). We have an icon file located in the icons folder with the name `Finder.png` that will be used as an icon for the application bookmark.

Example 3-3 JavaScript to add a custom icon

```
<link rel="apple-touch-icon" href="icons/Finder.png"/>
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

3. You can also expand **HATSiPhoneApp** → **Web Content** → **Common** → **Stylesheets** and double-click **whitetheme.** This will open up the `whitetheme.css` file in a CSS editor. You can change the application template here.

   Let us change the background color of the page from white to olive, as shown in Example 3-4.

Example 3-4 Changing the background color and font size

```
BODY {
   background-color: olive;
   color: black;
   font-family:  arial, tahoma, helvetica, sans-serif;
   /* For accessibility compliance: remove the following line */
   font-size: 10pt;
}
```

3.3.4 Recording macros

In our example we recorded two macros:

SignOn This macro will take us directly to the authentication panel.

CustInq This macro will take us to the application transaction panel once authentication is successful.

The steps are as follows:

1. You can record a macro with the Host Terminal window. To create a macro, click the **Record Macro** icon from the Host Terminal toolbar shown in Figure 3-14 on page 35.

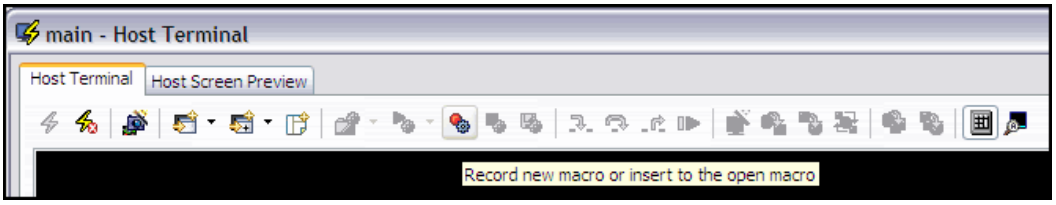


Figure 3-14 - Recording a new macro

2. You will see a pop-up window asking for the macro name; enter `SignOn` as the name of the macro, as shown in Figure 3-15.

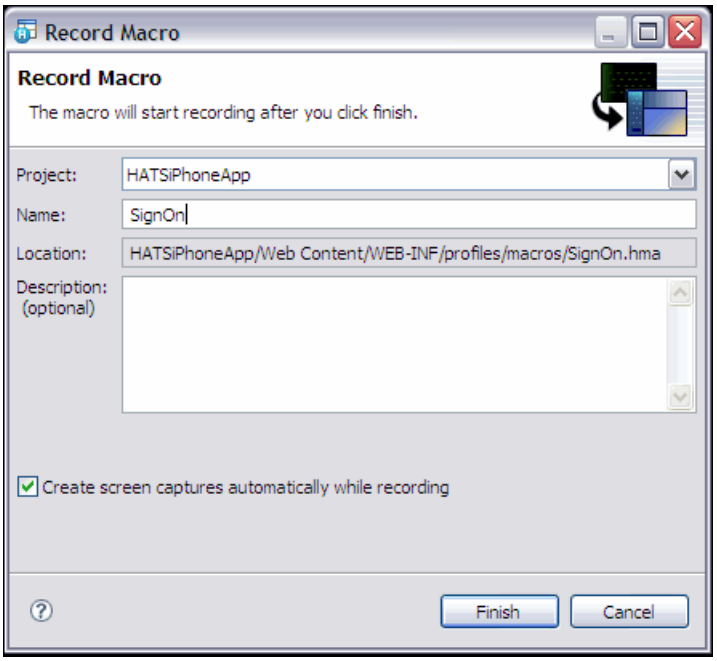


Figure 3-15 Naming a new macro

Make sure that the check box **Create screen captures automatically while recording** is selected.

3. Click **Finish.**

You will see the **Define Screen Recognition Criteria** window as shown in Figure 3-16 on page 36.

Note: The following steps depend on how your logon panel looks; you may have to perform these steps slightly differently, depending on the layout of your panel.

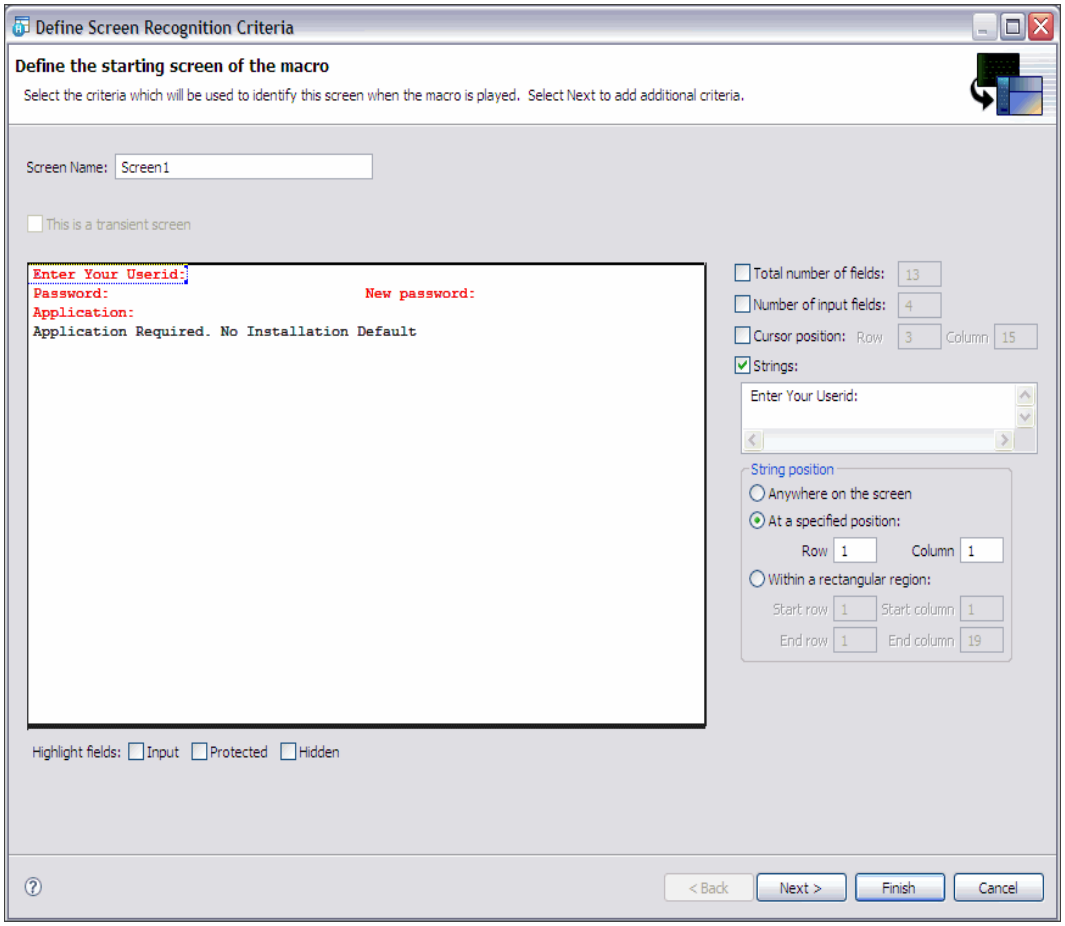


Figure 3-16 Define Screen Recognition Criteria

4. We selected the area around **Enter Your Userid:** which was surrounded consequently by a yellow rectangular box as shown in Figure 3-16.
5. Click **Finish.**
6. Now in the Host Terminal panel we entered `citso01` in the Application field and pressed the Control key. In our case this is the name of the CICS region. This took us to the next panel, and the Macro Navigator in the Host Terminal panel started to look as shown in Figure 3-17.

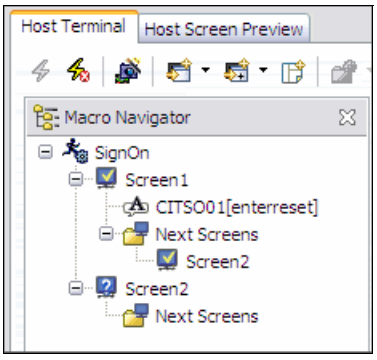


Figure 3-17 Macro Navigator

7. We then clicked the Stop Macro icon on the Host Terminal's toolbar and defined panel recognition criteria from the pop-up window as shown in Figure 3-18.

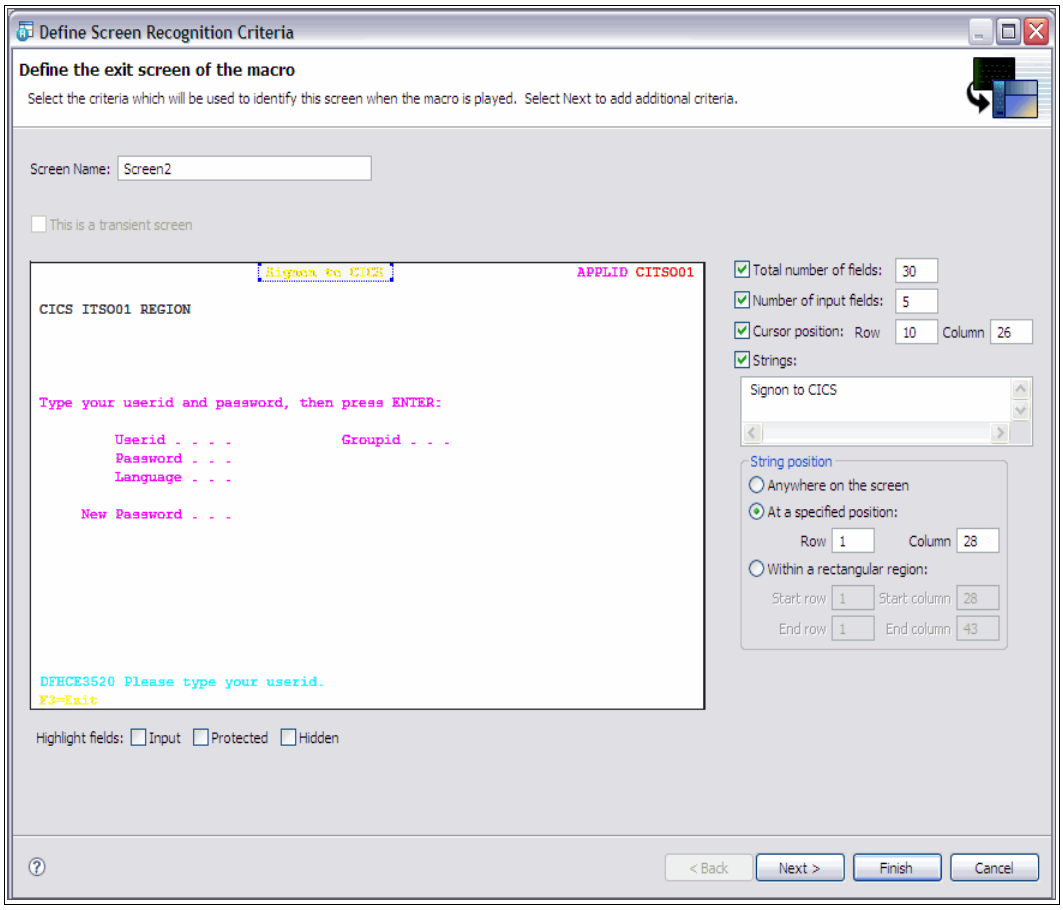


Figure 3-18 Define Screen Recognition Criteria for screen 2

8. We clicked **Finish** again.

   After these steps the macro was successfully recorded and now appears in the macros folder of the HATS Projects view of the workbench.

9. We double-clicked the **SignOn** macro in the HATS Projects view and it opened up the recorded macro with the complete panel flow shown in the macro editor, as shown in Figure 3-19.

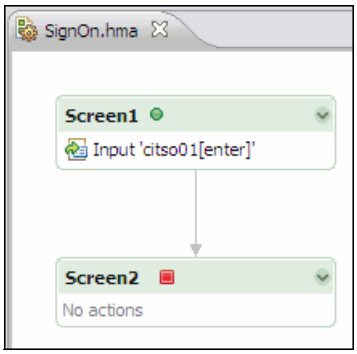


Figure 3-19 - SignOn macro in macro editor

10. Similarly, we recorded a second macro called CustInq that will take us from this panel to the next panel of entering a customer ID. After successfully recording the macro it will look in the macro editor as shown in Figure 3-20.

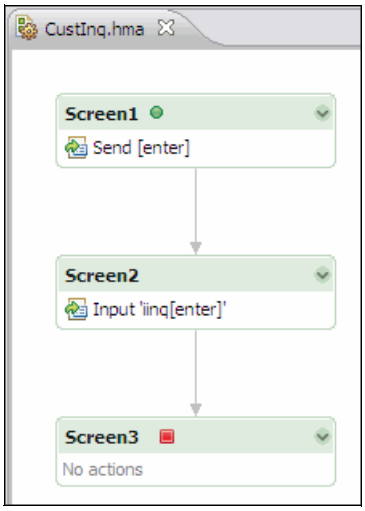


Figure 3-20 CustInq macro in macro editor

11. Once the macro is recorded it can be played from the Host Terminal panel toolbar by selecting a macro name from the drop-down menu of the Play Macro icon in the toolbar as shown in Figure 3-21.

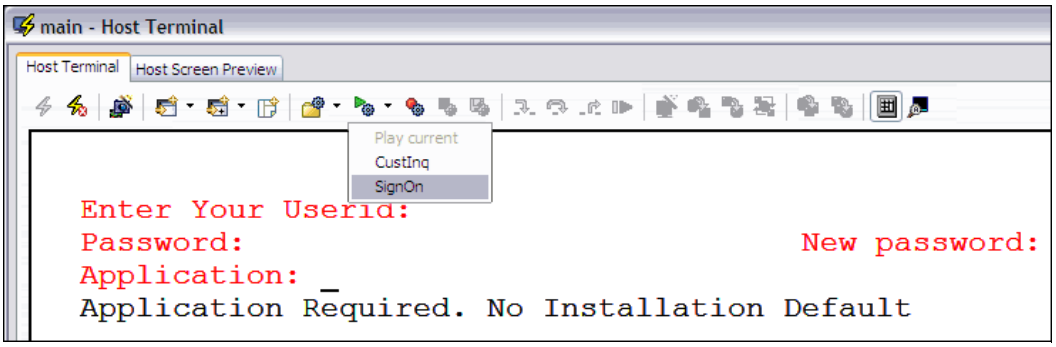


Figure 3-21 Playing a recorded macro

3.3.5 Customizing panels for Apple iPhone

Once the macros are recorded, you can customize the panels to be displayed on the iPhone panel as a graphical user interface. This is how we proceeded:

1. We browsed to the folder **Screen Captures** in the HATS Projects view and found subfolders with the names of the recorded macros, being SignOn and CustInq. In these subfolders there are panel captures associated with each of these macros.
2. We expanded these folders and saw the panel captures taken during the recording of the macros, as shown in Figure 3-22 on page 39.

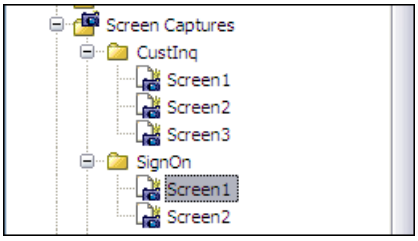


Figure 3-22 Screen captures taken for the macros

3. We double-clicked **SignOn → Screen2,** which opened Screen2 in the Host Screen Editor as shown in Figure 3-23. Now we can associate a panel customization.
4. We selected **New Screen Customization** from the Host Screen Editor's toolbar.

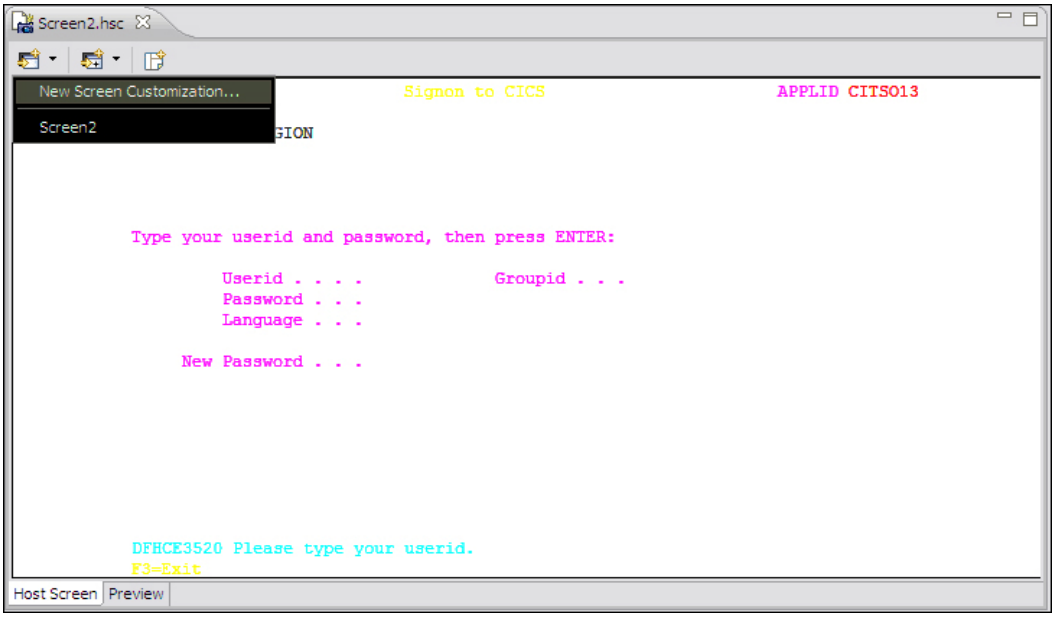


Figure 3-23 Screen customization

5. We specified a name in the popup window and clicked **Next**.
6. We defined the screen recognition criteria in the next panel and clicked **Finish.**

We then saw the pop-up dialog shown in Figure 3-24 on page 40.

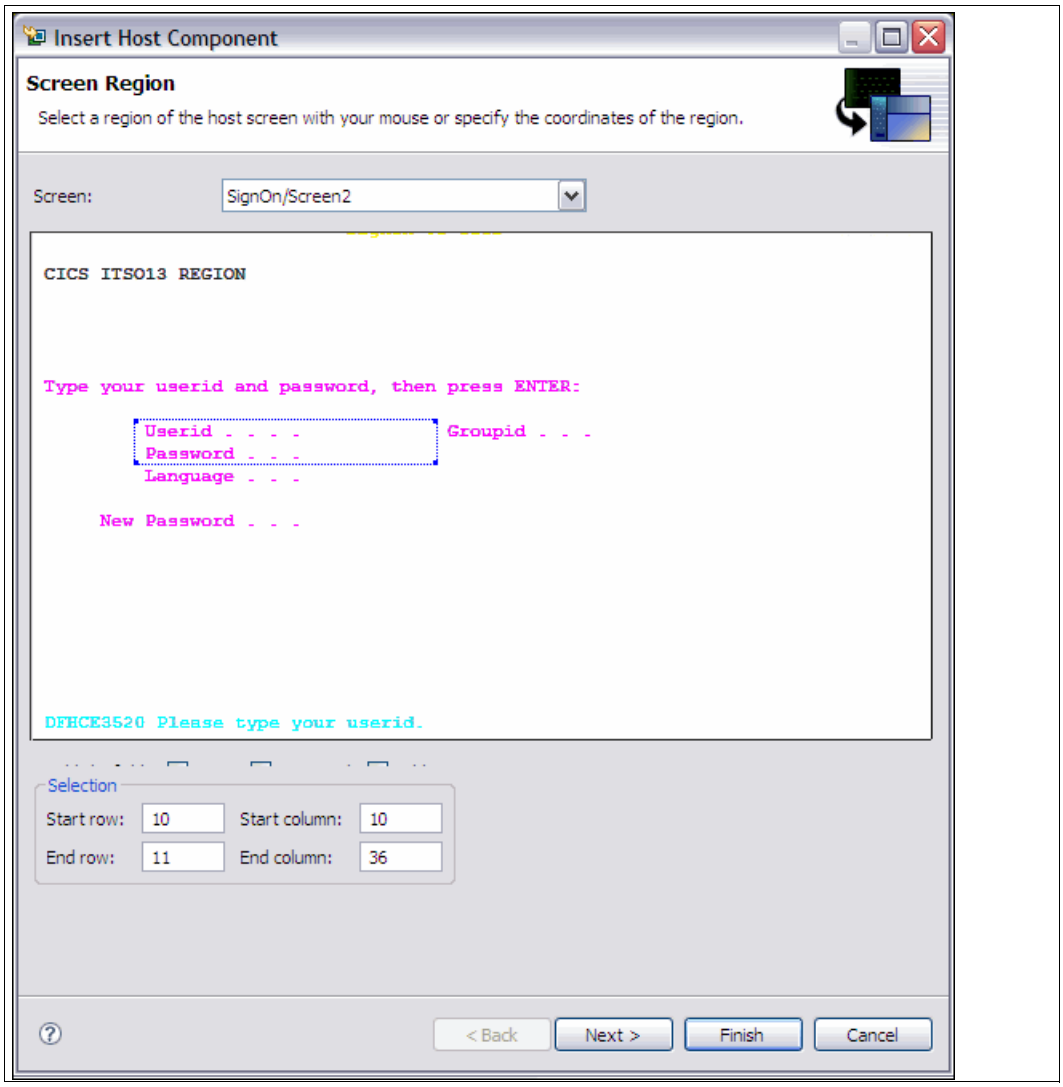


Figure 3-24 Customizing the panel area

7. Now we selected any area that we wanted to be customized for displaying it on the panel. This is done by selecting the region with a rectangular yellow box, as shown in Figure 3-24. We clicked **Next.**
8. We would like to show the selected region as a table, so we selected **Table (field)** from the Components table, as shown in Figure 3-25 on page 41. We then clicked **Finish.**

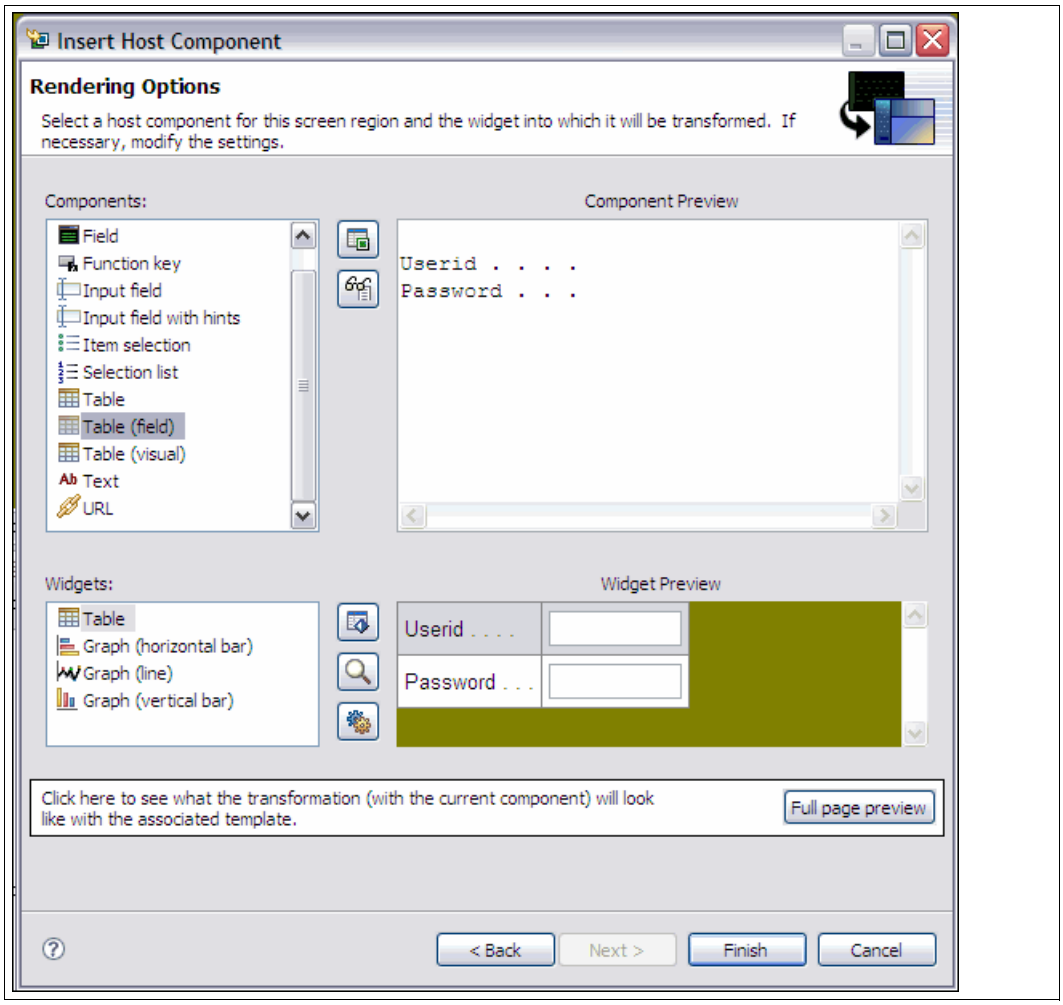


Figure 3-25 Rendering Options

9. We expanded the **Web Content** → **Transformations** folder in the **HATS Projects** view and noticed a new page with the name Screen2 added in the tree. We double-clicked it and the JSP editor opened with the Screen2.jsp page, as shown in Figure 3-26.

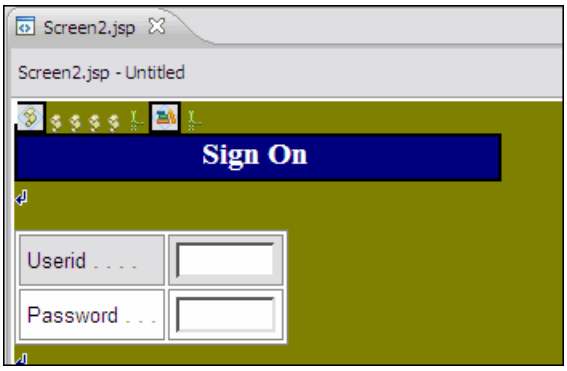


Figure 3-26 screen2.jsp in JSP editor

Similarly, we can also customize other panels associated with the CustInq macro.

10. We right-clicked on the Screen2.jsp page in the editor and selected **HATS Tools** → **Insert Macro Key** from the pop-up context menu. A pop-up window, as shown in Figure 3-27, was displayed.

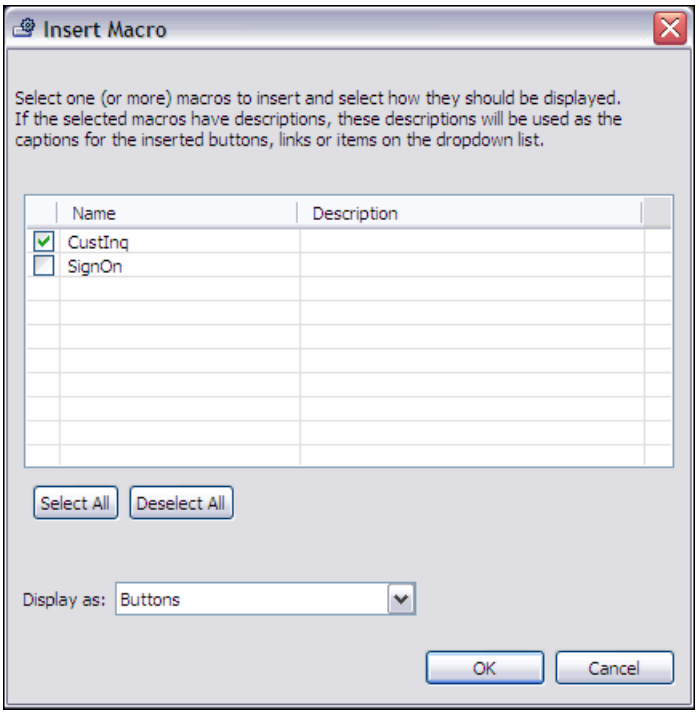


Figure 3-27 Insert Macro Key

11. We selected the **CustInq** macro in the popup window and also chose **Buttons** as the value for Display as: in the drop-down list. We clicked **OK**.

    This will insert a button on the JSP page. We specified it's label as **Login**.

At this point all tasks related to customizing the screens are completed. In the next section we will show how to test the application.

3.4 Creating a WAS connection for testing

Now, the application can be tested locally on a WAS server. Many of the Eclipse-based IBM development tools have an option to be installed with an embedded WAS.

1. Right click in the Server console panel area and select **New** → **Server**, as shown in Figure 3-28.

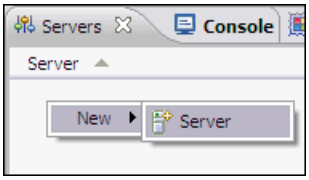


Figure 3-28 Creating a WAS connection

2. In the New Server panel, make sure you select the desired level of WAS server. Our example has been tested on a WebSphere Application Server Version 7.0 server. See Figure 3-34.

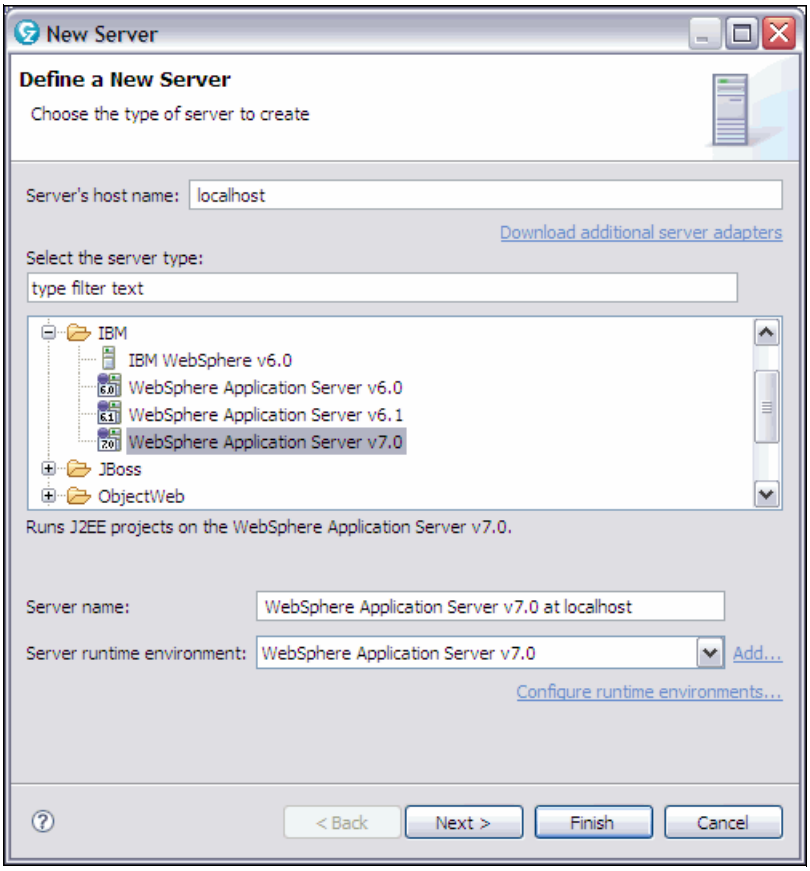


Figure 3-29 Defining a new WebSphere Application Server

3. Click **Next** and Figure 3-30 on page 44 is displayed. Here you can either manually or automatically determine connection actions. We chose **Manually provide connector settings**.

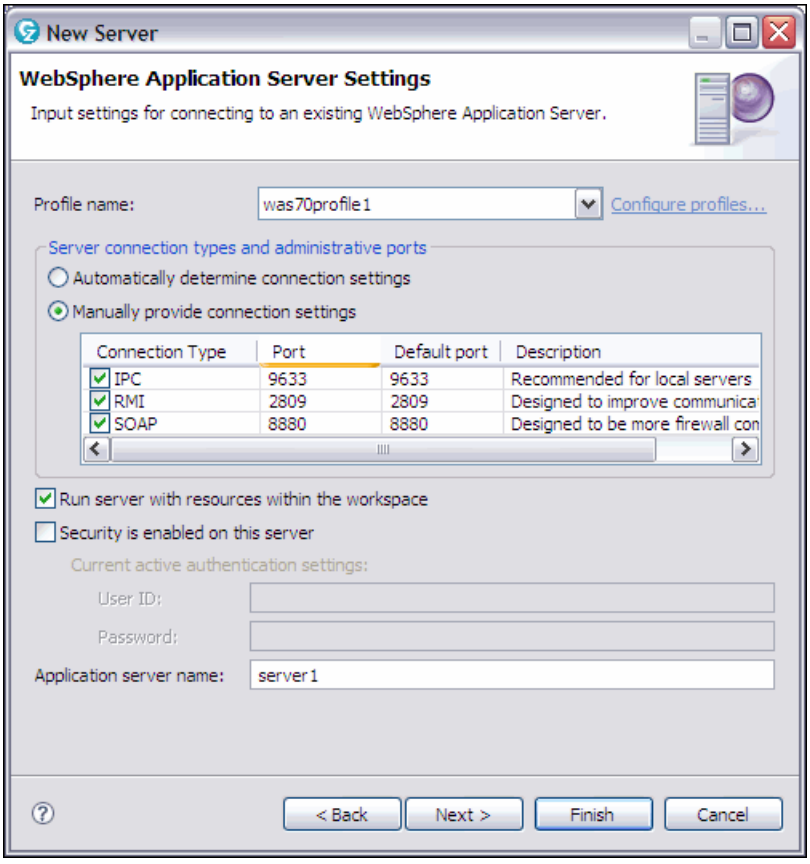


Figure 3-30 Input settings to connect to an existing WebSphere Application Server

4. Click **Next**. You will now be presented with the WebSphere Application Server Console shown in Figure 3-31. Note that the server state is Stopped.

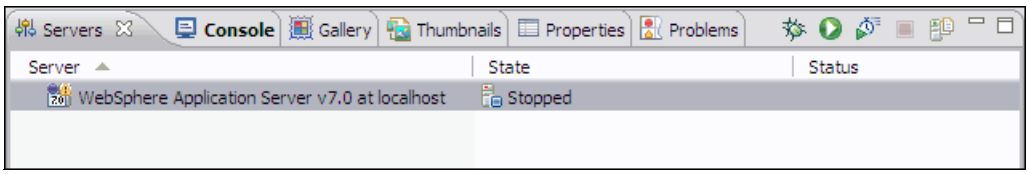


Figure 3-31 WebSphere Application Server console

5. Figure 3-32 shows the WebSphere Application Server after it has been started by right-clicking it and selecting **Start**.

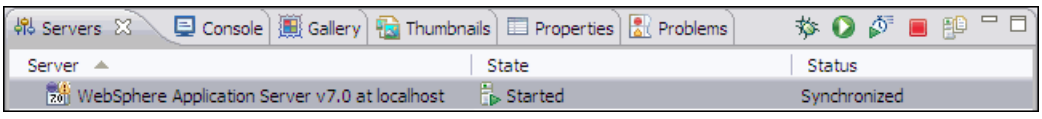


Figure 3-32 WebSphere Application Server started

6. Right - click the HATSiPhoneApp project and select **Run on server** as shown in Figure 3-33 on page 45.

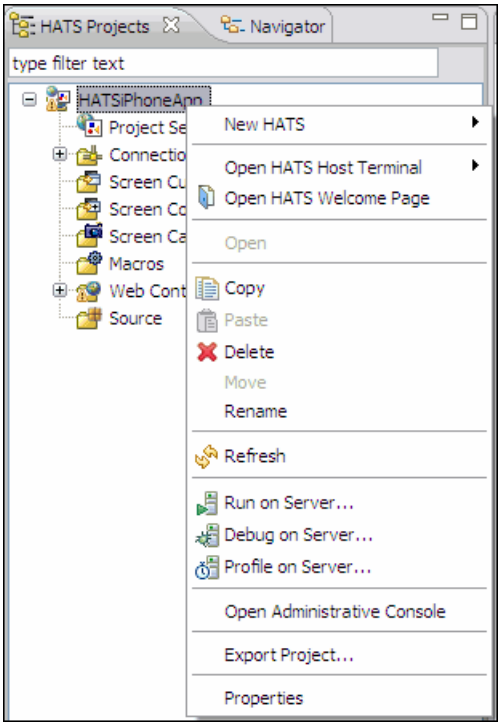


Figure 3-33 Running the application on the server

7. Choose the server that you started in step 5 on page 44, as shown in Figure 3-34, and select **Next**.

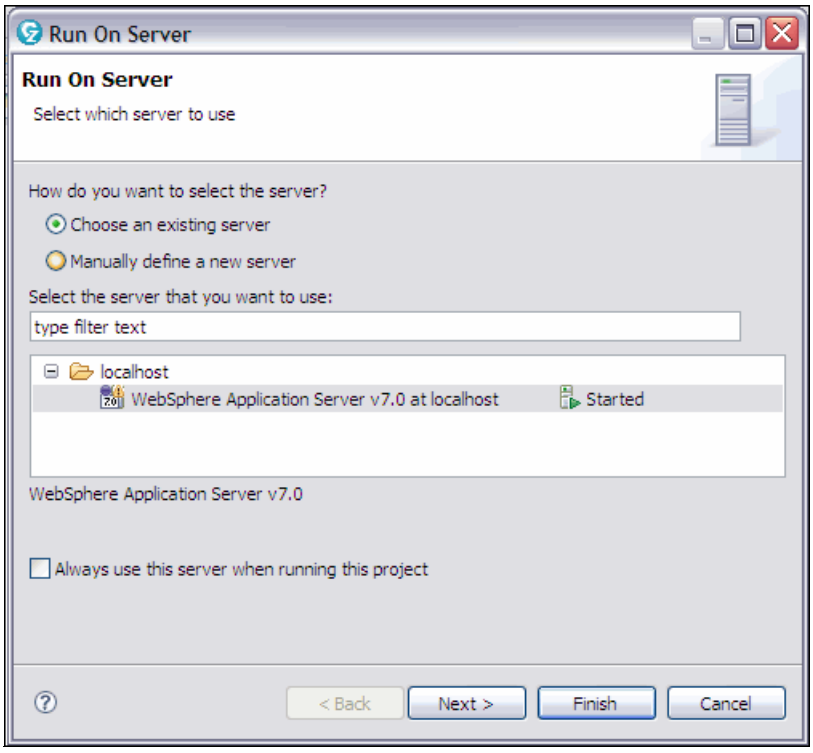


Figure 3-34 Selecting the server

8. Make sure your project appears in the Configured projects panel. If it does not, select it and then click **Add** (Figure 3-35).

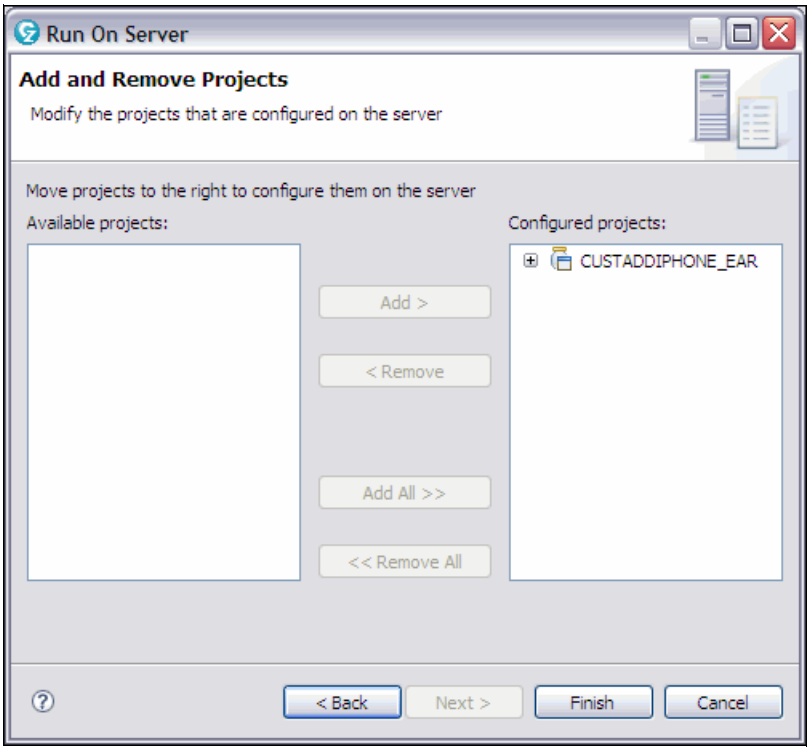


Figure 3-35 Adding your project

9. Click **Finish.**

Now the application is deployed to a local WebSphere Application Server on a Windows workstation.

3.5 Testing the HATS application

Attention: The next steps assume you have access with your smartphone to the Internet and access through the network to the Windows server that the WebSphere Application Server is running on.

If you are testing this scenario on a Windows server with a corporate intranet, you may probably need to set up your smartphone with VPN access into the corporate network in order to be able to reach your Windows server.

To test the application with a smartphone, proceed as follows:

1. On the smartphone device, open a Web browser and type in the address of the application. In our case it is `http://9.57.139.23:9081/iPhoneApp`, as shown in Figure 3-36 on page 47. In your case, substitute the IP address and port number with your own values. You can also use a host name if you are using a DNS.

   You will notice the Sign on panel of the application being loaded into the browser.

Figure 3-36 Authentication panel

2. Insert the user ID and password for accessing the CICS transaction and press the **Login** button. You will have to use the on-screen keyboard available on the Apple iPhone. See Figure 3-37 on page 48.

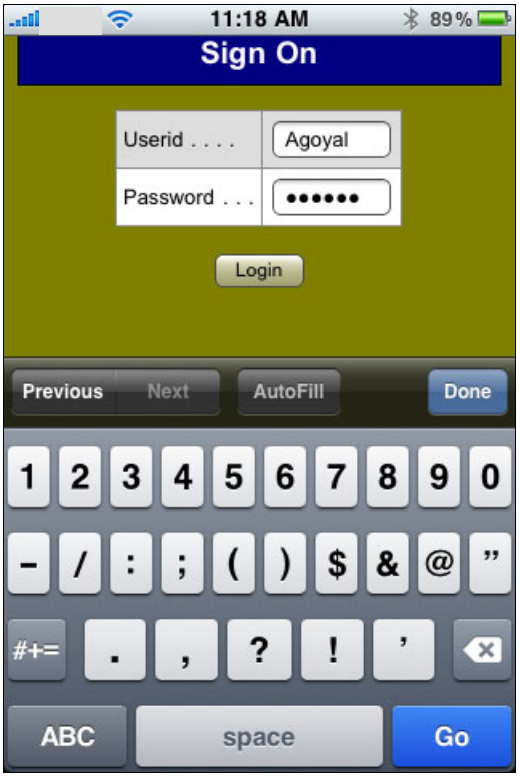


Figure 3-37 Authentication

3. This will take you to the next panel of the CICS transaction Customer Inquiry, where you can insert a customer ID and fetch the details on the iPhone panel. See Figure 3-38.

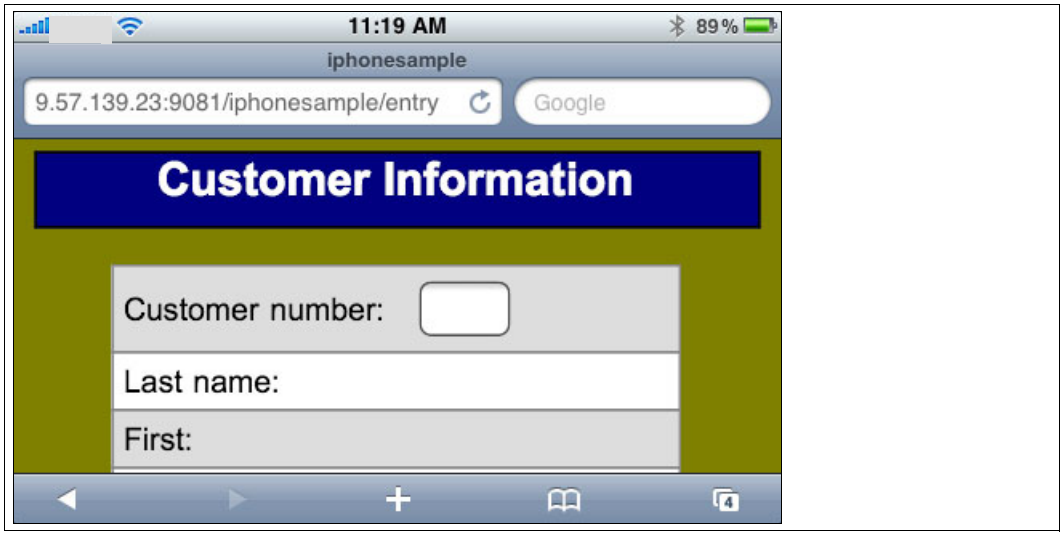


Figure 3-38 Customer information panel

You can see the output results being pulled from CICS and successful working of the HATS application. See Figure 3-39 on page 49.

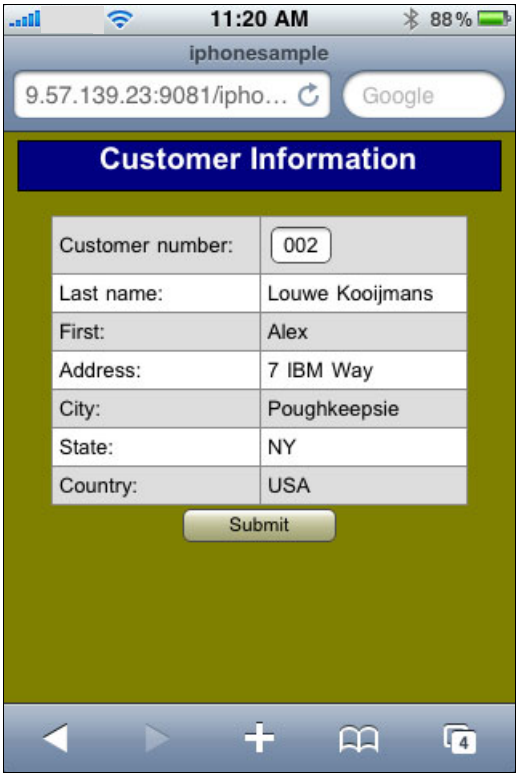


Figure 3-39 Customer information panel filled in

Congratulations! You have successfully extended a Host CICS transaction to Apple iPhone with the help of a HATS Web application.

3.6 Deployment to WebSphere on z/OS

The HATS application can also be tested and used in WebSphere Application Server on z/OS. Refer to Appendix C, “Deployment to WebSphere Application Server on z/OS” on page 197 for details.

4

Accessing CICS in Atom feeds format using WebSphere sMash

In this chapter we describe the architecture and the steps to be performed in order to access CICS data via Atom feeds. The middle-tier application server used in this scenario is based on the WebSphere sMash platform. It can be downloaded for free from the Project Zero Web site at:

http://www.projectzero.org/

It can be installed on any personal computer.

Instead of developing a brand new WebSphere sMash application, we chose to modify an existing sample provided by the product itself. Many samples are included with the product that cover most of the possible needs for any programmer. Modifying an existing sample is a common way to develop an application with WebSphere sMash. One of the main advantages of WebSphere sMash is enabling low-skilled application developers to create professional applications in a short time.

Some prerequisites are required for the implementation of this scenario. It is assumed that mainframe components are activated and ready to be invoked. Any existing customer application can be accessed in the way we will describe. To avoid any problems, the source of a simple COBOL CICS/VSAM application is included in the sample, so it is easier to activate everything, even if no suitable application is available.

© Copyright IBM Corp. 2010. All rights reserved.

4.1 High-level architecture

Figure 4-1 shows the architecture we are going to implement. The protocol of communication between the iPhone and WebSphere sMash application server is HTTP. The user of this application fills out a FORM to pass user data to the application, then a call to the Atom service is done. After that call, CICS data in Atom format is returned to the application.

At this point, the application determines what to do next. It is possible to call another service (internal or external, such as GoogleMaps), or return the data directly to the caller (in this case, the smartphone).

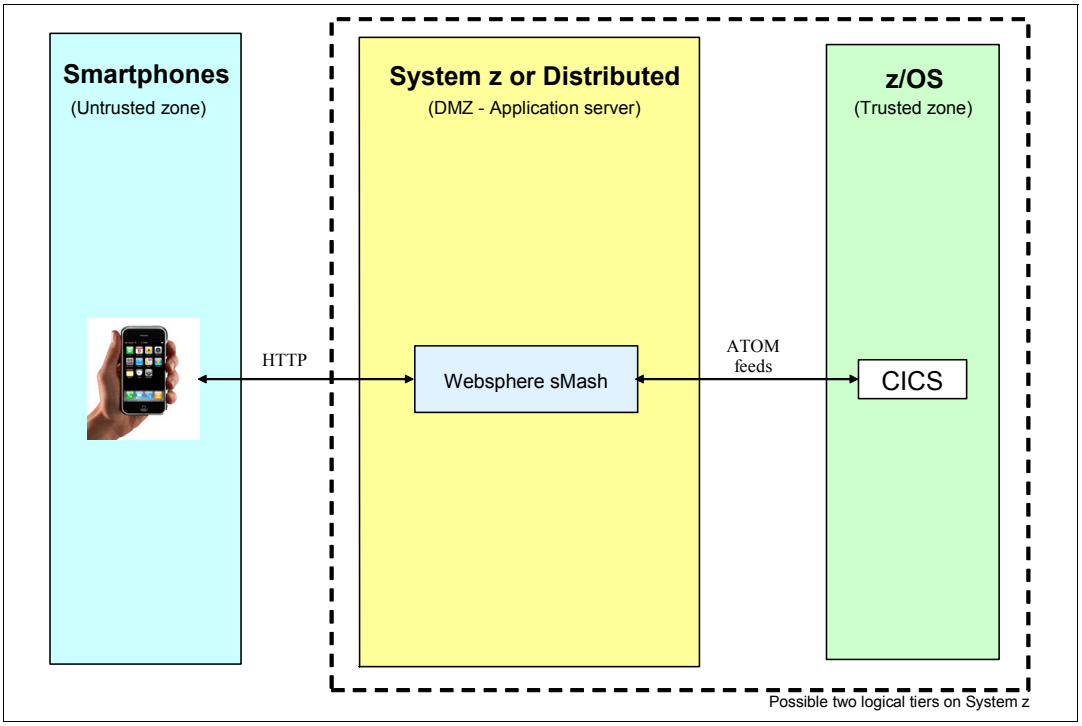


Figure 4-1 Solution architecture - Accessing a CICS application on z/OS using WebSphere sMash and ATOM feeds

4.2 Software used in this scenario

The prerequisite software used for the development of this scenario was:

- Windows, Linux, or Mac operating system
- Java SE Development Kit (JDK) (32-bit only)
- Firefox Web Browser

And then, of course, you will need the WebSphere Smash environment, which we discuss in 4.3.1, “Step 1: Installing WebSphere sMash and running the IVP” on page 53, as well as a CICS back-end environment on z/OS, configured for using Atom feeds and with a pre-installed application of your choice.

4.3 Steps followed for implementation

In describing the steps required to set up the scenario, we assumed that an Atom CICS feed service is already in place. In order to set up the Atom feeds service, CICS TS Version 3 (or later) is needed. Support to the Atom feeds in CICS TS V3.1 and V3.2 is provided by means of the CA8K support pack. Starting with CICS V4.1, Atom feeds support is integrated into the CICS product.

4.3.1 Step 1: Installing WebSphere sMash and running the IVP

The initial step will be to set up the application server tier where our application will run. This is a basic scenario and we want to keep it as simple as possible, so we'll use WebSphere sMash as the application server platform. WebSphere sMash can be downloaded from the official Project Zero Web site at:

http://www.projectzero.org

Project Zero is a technology incubator project centered around agile development and the next generation of dynamic Web applications. The project introduces a simple environment for creating, assembling and running applications based on popular Web technologies. This environment includes a scripting runtime for Groovy and PHP with application programming interfaces optimized for producing REST-style services, integration mashups, and rich Web interfaces.[1]

When accessing the Project Zero Web site, you will see the home page shown in Figure 4-2 on page 54.

[1] Excerpt from http://www.projectzero.org

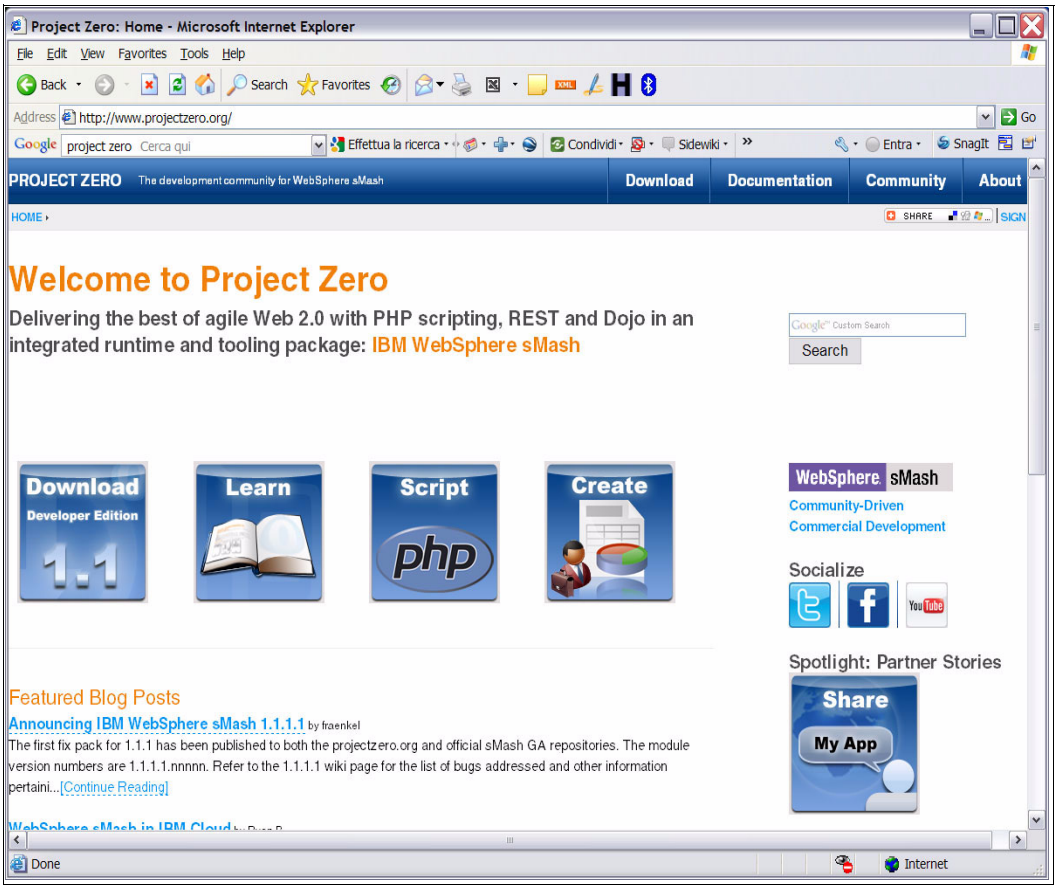


Figure 4-2 Project Zero Web site: initial page

It is possible to download a complete application server environment from this Web site. Download the latest free version of WebSphere sMash. It can be used for developing applications, or it can be used as a runtime for executing applications prepared by others.

To download, go to the following address, as shown in Figure 4-3 on page 55:

`http://www.projectzero.org/download/`

First you have to download the *Command Line Interface (CLI)* and then start the *Application Builder Interface*. There are instructions on the Web site on how to do that.

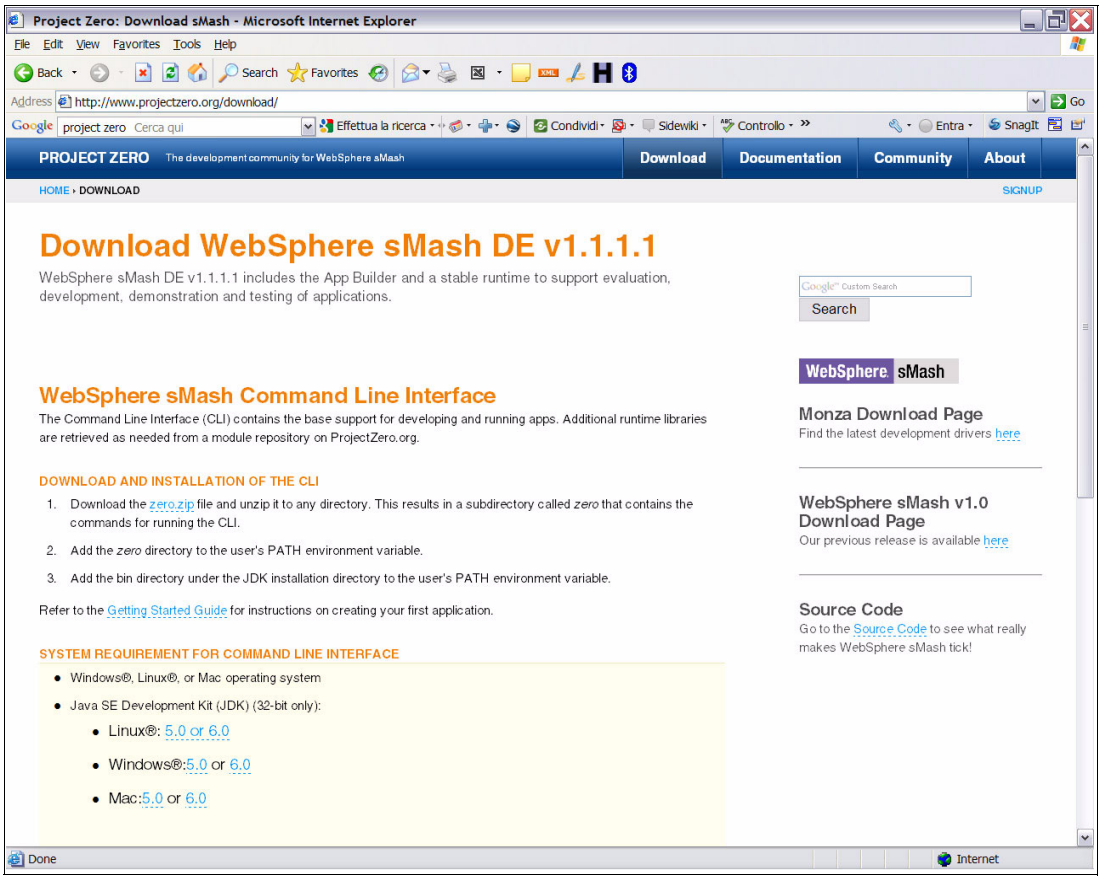


Figure 4-3 Project Zero Web site: download page

After product installation, it is possible to download a sample application to test whether you have performed the required actions in the correct way. There are many samples available from the sample download page, found at:

`http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/samples.doc/SamplesOverview.html`

To do this, download a *hello world* type application, as you only need to test the installation, not application functionality. On the sample download page, there is a small application called Hello Dojo. It is perfect to cover our needs. Here are the steps you would perform in order to deploy this small application:

1. Start the *Application Builder Interface*, from the Project Zero directory. In our example, this directory is `c:\zero`. You would then type the following command:

   ```
   appbuilder open
   ```

   With the first execution of the command, you will see the log entries shown in Example 4-1 on page 56, and a new Firefox Web browser session will start and open to:

   `http://localhost:8070/`

   Subsequent executions of the command will only show the lines in bold as indicated in Example 4-1 on page 56 and open the browser.

Example 4-1 Starting the Appbuilder

```
-------------------------------------------------------
Creating the AppBuilder application.
This step may take a while.
-------------------------------------------------------
CWPZT0849I: The new module will be created from module
zero:zero.application.tem
plate:latest.revision, located at F:\WebSphere
sMash\zero\zero-repository\stable
\modules\zero\zero.application.template\zips\zero.application.template-1.1.1.1.
3
0533.zip
CWPZT0840I: Module appbuilder.bootstrap successfully created
CWPZT0600I: Command update was successful
CWPZT0600I: Command modulegroup create was successful
CWPZT0901I: The following module(s) are not currently in the local repository:
        zero:zero.cli.tasks:[1.0.0.0,)
CWPZT0902I: Trying to locate the module(s) using one of the configured remote
repositories
.....
CWPZT0545I: Retrieving zero.application.template-1.1.1.3.31044.zip from host
http://www.projectzero.org/zero/monza.dev/latest/repo/
CWPZT0531I: Module List :
zero:zero.application.template:1.1.1.3.31044
CWPZT0849I: The new module will be created from module
zero:zero.application.tem
plate:latest.revision, located at F:\WebSphere
sMash\zero\zero-repository\experimental\modules\zero\zero.application.template\
zips\zero.application.template-1.1.1.3.31044.zip
...
CWPZT0545I: Retrieving zero.application.template-1.1.1.3.31044.zip from host
htt
p://www.projectzero.org/zero/monza.dev/latest/repo/
CWPZT0531I: Module List :
zero:zero.application.template:1.1.1.3.31044
CWPZT0849I: The new module will be created from module
zero:zero.application.tem
plate:latest.revision, located at F:\WebSphere
sMash\zero\zero-repository\experi
mental\modules\zero\zero.application.template\zips\zero.application.template-1.
1
.1.3.31044.zip
CWPZT0840I: Module appbuilder successfully created
Application started and servicing requests at http://localhost:8070/
CWPZT0600I: Command start was successful
```

The initial **My application** page will be displayed, as shown in Figure 4-4 on page 57. This is the initial page, where in the future you will be able to see the list of all installed applications.

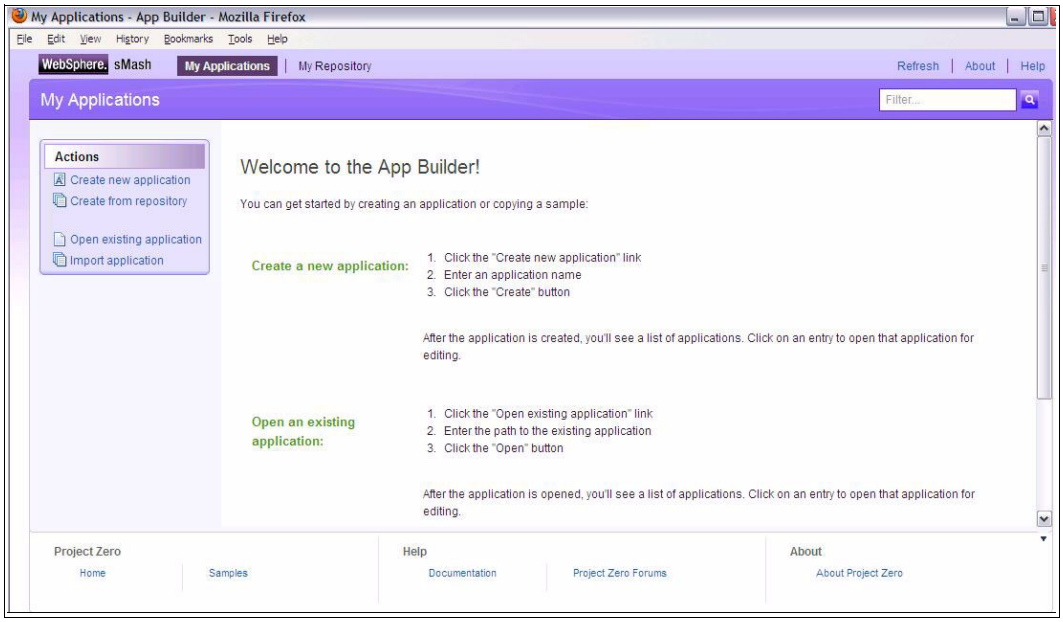


Figure 4-4 AppBuilder home page

2. To install a sample package from the repository, you have to click in the top left side, on the **Create from repository** link. A pop-up window will appear, and it is possible to insert a filter in order to reduce the amount of packages listed. Using “hello” as filter will help you to find the zero.hellodojo.demo application.
3. Once the application is selected, click **Create** on the top right of the panel. This will start the package installation as shown in Figure 4-5 on page 58.

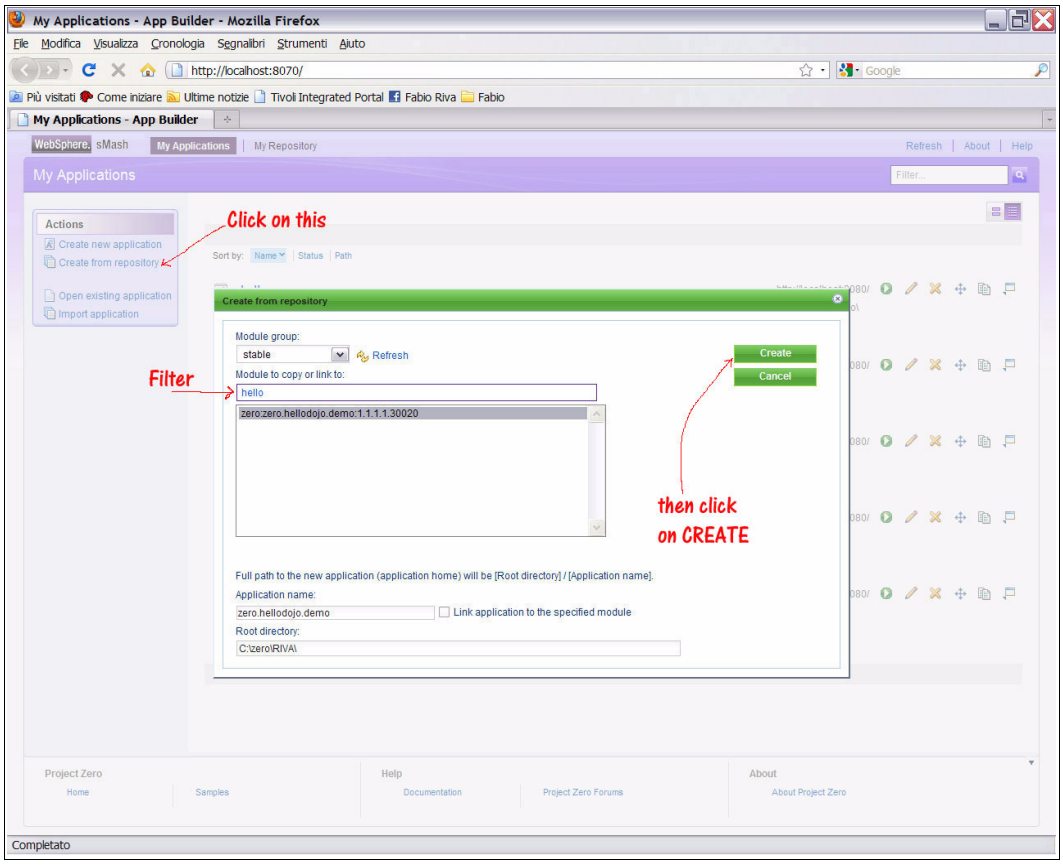


Figure 4-5 Creating a local copy of the Hello Dojo package from the repository

Once installed, the *Hello Dojo* demo application appears in the application list.

From this page you can control all the installed applications; you can start or stop them, delete, package, and so on. You have complete control over the workload on the application server. More than one application can be started at a time.

4. In order to start an application, click the green arrow beside it. The application will start in a few minutes and it will be accessible by the address link shown near the green arrow. The green arrow becomes a small red square, indicating that the application is running. If you want to stop the application, click that square (see Figure 4-6 on page 59).

Note: It should be noted that if you did not stop zero with the command **`zero stop`**, starting your application will fail with the message:

```
CWPZC8017E: ZSO unable to bind to port 8080.  Port may already be in use.
```

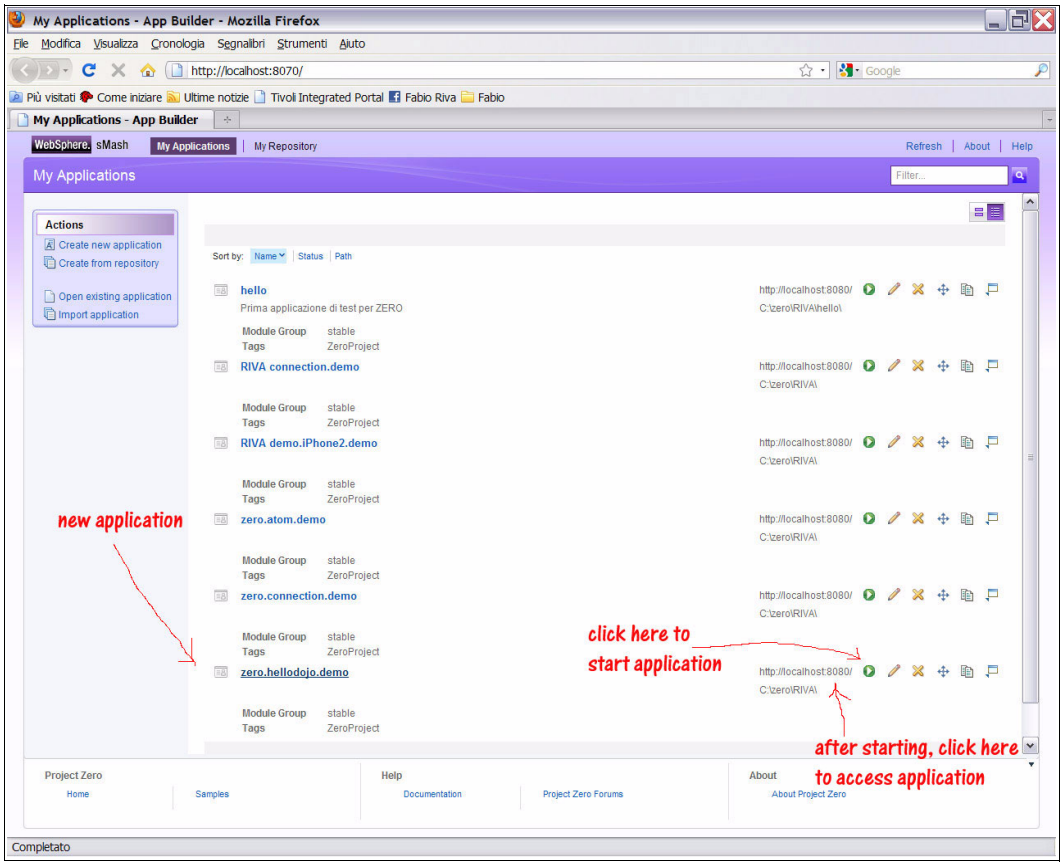


Figure 4-6 My applications page: starting the new application

It is possible to access applications using the address link provided by the WebSphere sMash *My Application* Web page. Once clicked, a new Firefox browser session is started and the initial application Web page is shown.

We chose to download a very simple application from the repository, so it would be easy to test. Our intention was to verify our sMash installation, so we just needed a simple application.

To test the application, type a name in the upper box (in our example, “Chiara”) and the statement “Hello Chiara” will appear in the History box (see Figure 4-7 on page 60).

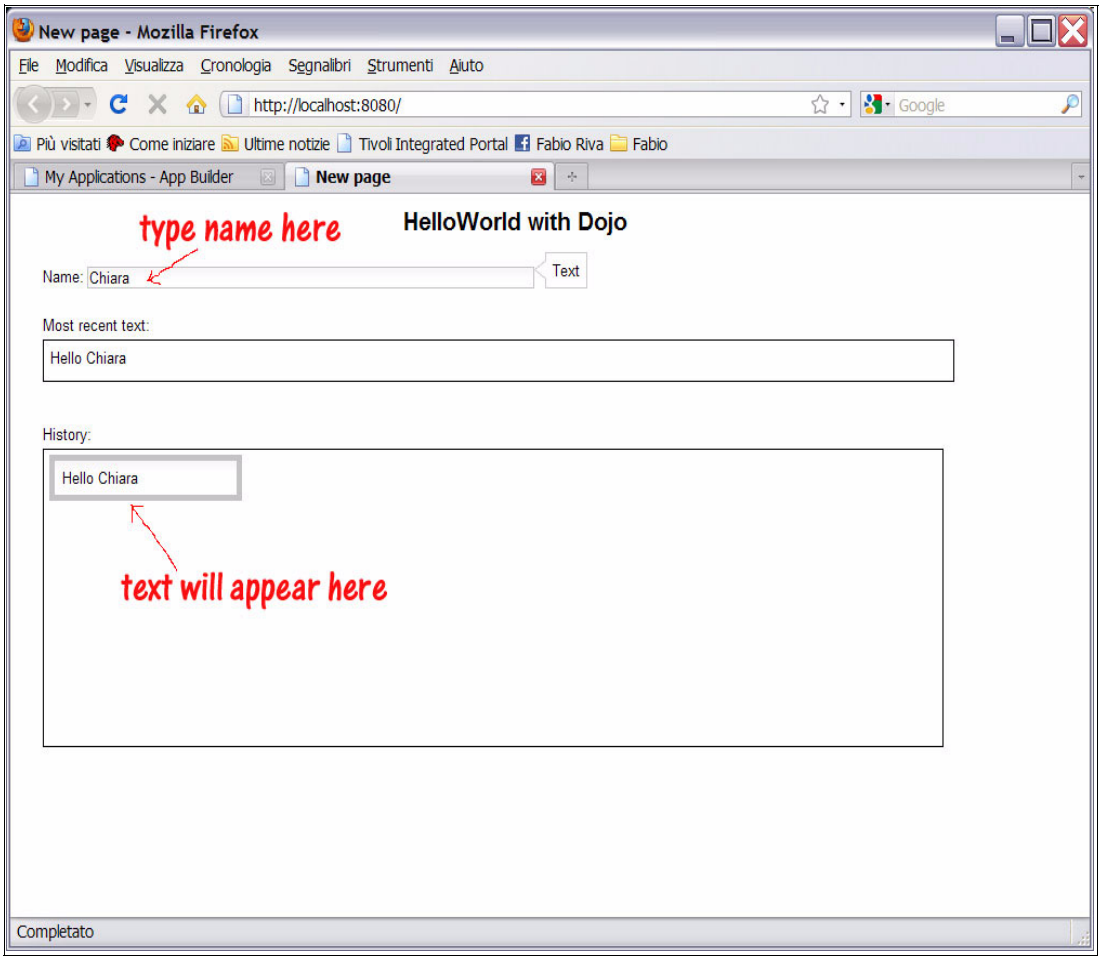


Figure 4-7 Testing the new Hello Dojo application

4.3.2 Step 2: Downloading and modifying a sample application from the sMash repository

Once you set up the infrastructure to develop an application, you are ready to work with it. Again, instead of writing a brand new application, in our example we downloaded a sample from the repository and modified it.

Looking in the projectzero.org Web pages, we found a link titled “Overview of tutorials, samples, and demos” at:

`http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/samples.doc/SamplesOverview.html`

This Web page can also be reached from the projectzero.org Web site and by navigating to the documentation page.

The example we are using is called Connection API. It is described as: “Contains example uses of the server-side Connection API, such as invoking a REST service and sending an e-mail.”

The steps are as follows from now on:

1. The first step is to download the sample from the repository. You can do this by clicking on the top left side of the My Applications-App Builder page, reached from the “Create from repository” link. A pop-up window will appear. Insert a filter in order to reduce the amount of packages listed. You can use the words “connection.demo” as a filter, to simplify the search of the package. Some entries will probably be found, and you should choose the most recent one. In our case, this was zero:zero.connection.demo.1.1.1.1.29382.
2. After installation, click the application address to access the application. There are some actions that can be tested with this package. They are not real connections, they are just calls between programs provided with the package itself. You will need to modify them in order to access a real Atom feed service provided by CICS on z/OS.

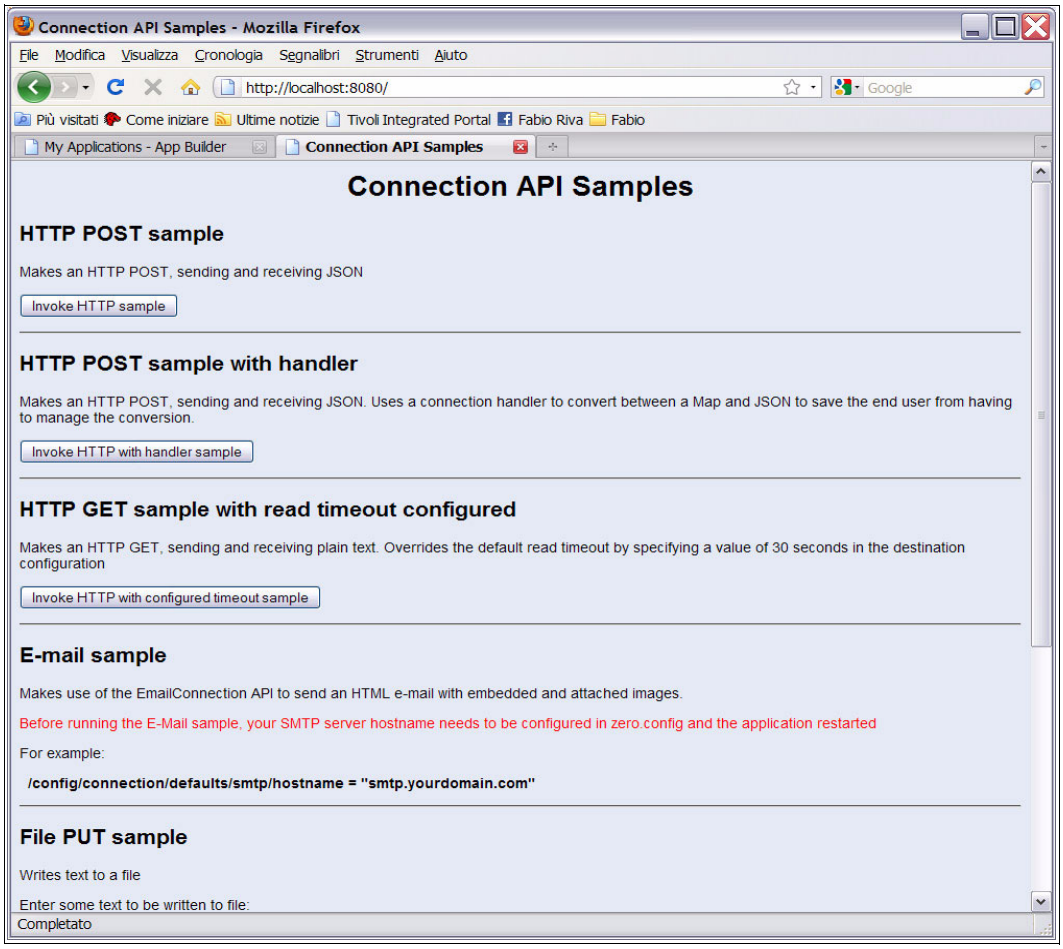


Figure 4-8 Executing the Connection API sample from repository

As shown in Figure 4-8, all the possible samples are listed. You will need to modify the third option “HTTP GET sample with read timeout configured”. We plan to:

- Modify the target address for the Atom feed request, pointing to CICS on z/OS
- Create a form to be filled in order to prepare the parameters for the request
- Add needed HTML instructions for the iPhone layout and usage
- Add a nice-to-have feature to our iPhone Web interface

In order to do this, you should create a copy before you start to modify the application. In this way you will have a reference to the initial status and you will be able to modify the

new application as much as you like. We chose a name for the new application. In the sample we are using, we adopted the prefix CLEAR.

3. To create a copy of the application, click the double page icon located on the row of the application on the “My Applications - App Builder” page and provide a name to the new entry. We chose to call it CLEAR.
4. Running the application, the initial HTML page that is shown is described in the `index.gt` file and can be found in your file system. In our file system, it was `C:\CLEAR\public`. The first modification will be to create a version of `index.gt` that will prepare a FORM in order to pass data to the httpWithTimeout.groovy program. This will be the actual interface to CICS. To edit this file:
   a. From the “My Applications - App Builder” page, click the pencil icon on the row of your CLEAR application.
   b. On the left navigator pane, in the search box, type `index.gt` to find the correct file.
   c. Click that file name, and on the right navigator pane you will see the raw HTML page exposed for editing.

   The HTML statements inserted on this page to support and manage the Web interface to the iPhone, the <meta> statements, are shown in Example 4-2.

Example 4-2 Sample HTML with statements for iPhone highlighted

```
<!--/*
 * ============================================================================
 * (C) Copyright IBM Corporation 2007, 2008.  All Rights Reserved.
 * index.gt
 * You may only copy, modify, and distribute these samples internally.
 * These samples have not been tested under all conditions and are provided
 * to you by IBM without obligation of support of any kind.
 *
 * IBM PROVIDES THESE SAMPLES "AS IS" SUBJECT TO ANY STATUTORY WARRANTIES THAT
 * CANNOT BE EXCLUDED. IBM MAKES NO WARRANTIES OR CONDITIONS, EITHER EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR
 * CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
 * NON-INFRINGEMENT REGARDING THESE SAMPLES OR TECHNICAL SUPPORT, IF ANY.
 * ============================================================================
 */ -->

<html>
<head>
<title>Getting Atom feeds data from CICS</title>
<meta name="viewport" content="user-scalable=no, width=device-width" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black" />
<link rel="apple-touch-startup-image" href="loading.png" />
<link rel="apple-touch-icon" href="iphone.jpg" />
</head>

<body style="font-family: arial; font-size:16px; background-color:#e4eaf4"
background="images/z10.gif" >

<h1><center>Test Query of CICS Data</center></h1>
<hr>
<h2>Query on CICS Data</h2>
<h3>Query</h3>
```

```
<p>This query calls CICS zOS via Atom feeds and returns mainframe data </p>
<form id="http1" method="POST" action="http/httpWithTimeout.groovy">
<b> Select Product Code:</b>
<SELECT>
<OPTION selected value=”0100”>0100</OPTION>
<OPTION value=”0200” >0200</OPTION>
<OPTION value=”0300” >0300</OPTION>
<OPTION value=”0400” >0400</OPTION>
<OPTION value=”0500” >0500</OPTION>
</Select>
<input type="submit" name="invoke" value="Submit selection"/>
</p>
</form>
<br>
<hr>
</body>
</html>
```

The images referred to in the HTML should be placed in the `\CLEAR\public` or in the `\CLEAR\public\images` directory. In this example, you will notice a `z10.gif` file. This is just a background image, as shown in Figure 4-9.

Figure 4-9 z10™.gif file

5. After this modification, you can start the CLEAR application to see the new layout of the interface.

   Making a selection and clicking the **Submit selection** key will invoke the httpWithTimeout.groovy application. This is a dummy program in the repository sample; it will simply return a string with “HTTP (no handler) invocation successful. Message from dummy service was Hello Fred from dummy”.

In the next steps we show how to change the interface in order to call an Atom feed service in CICS on z/OS. The modifications are to the original httpWithTimeout.groovy program, found in the `/public/http` directory file structure. All the modifications made are shown in Example 4-3 and preceded with the // ITSO comment prior to the line of code that we modified.

- We modified the Internet address where the program will ask for Atom feeds (labeled as 1 in our figure). In the sample provided, a call to an internal program was done. We replaced it with a valid call to Atom service.
- We added an instruction in order to get the value of a parameter passed by the FORM we included before (labeled as 2). This is needed to call the Atom service, because it is the product code we are using for inquiry.
- We inserted some HTML instructions (using the <meta> tag) in order to adapt the Web page to the iPhone layout (labeled as 3). In addition to that, a loading image is provided to signal the running of the application while downloading the code.

Example 4-3 Adding HTML for iPhone layout, parameter from POST and pointer to CICS

```
/*
 * ============================================================================
 * (C) Copyright IBM Corporation 2007, 2008.  All Rights Reserved.
 * You may only copy, modify, and distribute these samples internally.
 * These samples have not been tested under all conditions and are provided
 * to you by IBM without obligation of support of any kind.
 * IBM PROVIDES THESE SAMPLES "AS IS" SUBJECT TO ANY STATUTORY WARRANTIES THAT
 * CANNOT BE EXCLUDED. IBM MAKES NO WARRANTIES OR CONDITIONS, EITHER EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR
 * CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
 * NON-INFRINGEMENT REGARDING THESE SAMPLES OR TECHNICAL SUPPORT, IF ANY.
 * ============================================================================
 */
package http;
import java.util.HashMap;
import zero.core.connection.Connection;
import zero.json.Json;

/**
*
 * This is a sample of a simple HTTP GET request using the Connection API.
 **/
def onPOST() {
   // ITSO
   def ret = '<html><head><title> CICS DATA</title>';
   ret = ret + '<meta name="viewport" content-user-scalable=no, width=device-width" />';
   ret = ret + '<meta name="apple-mobile-web-app -capable" content="yes" />';
   ret = ret + '<meta name="apple-mobile-web-app-status-bar-style" content="black" />';
   ret = ret + '<link rel="apple-touch-startup-image" href="loading.png" /></head>';

   // def ret = "HTTP (with timeout set) invocation successful";
   // ITSO
    def message=request.params.message[0];
    try {
      // Create a new Connection targetted at the http address we wish to post data to.
      // Change the ip address to one of your own
      // ITSO (Changed IP address to z/OS server with CICS)
      def conn = new Connection("http://9.71.198.33:38083/atom/r/atomfriva?s=" + message);
```

HTML statements for iPhone layout

Instruction to get parameter from POST

Pointer to Atom service in CICS on z/OS

```
    // Test your code using the following line by commenting the above line and uncommenting
    // the next line
    // def conn = new Connection("http://localhost:8080/http/alt2/dummyAlt2.groovy");

    // We are performing a GET
    conn.setOperation(Connection.Operation.GET);

    // Get the response from the http service. Calling send causes the request to
    // be sent and returns the object from which the response can be obtained.
    def resp = conn.send();

    // Get the data returned from the service in the form of a String
    def str = resp.getResponseBodyAsString();
    // Additional code will be placed here!
    // End of additional code
    ret = ret + ". Data from dummy service was <b>" + str + "</b>";

    } catch (Exception e) {
    ret = "HTTP (with timeout set) invocation failed with exception " + e;
  }
  def html = invokeMethod("helper.groovy", "getResponseHTML", ret);
  println(html);
}
```

Some modifications are still missing. The sample code from the repository manages the Atom feed data from the server as a string. Some parsing activity is needed to prepare the output in HTML format, and work on that next.

Many z/OS system programmers develop small programs in REXX code to satisfy their needs. REXX is very popular amongst mainframe people and it is widely used. In the modifications we are going to apply, there is more REXX programming than Java/Groovy programming, and this shows that just about anyone can download a sample from the WebSphere sMash repository and modify it!

To add the next set of instructions, refer to Example 4-3 on page 64, where there is a bold line indicating ”// Additional code will be placed here!”.

The code you need to insert is labeled as follows in Example 4-4 on page 66:

- The code labeled “A” will parse the string and will resolve the variables to the correct values.
- The instructions labeled “B” create HTML statements with the correct values coming from the Atom feed request. This is done for proper formatting. If you want to receive Atom feed data as a string, you do not have to perform this modification.

Example 4-4 Other modifications to the sample

```
// ITSO
        def codpro = '-';
        def strarray = str.split("<codpro>");
        def strarray2 = strarray[1].split("</codpro>");
        codpro = strarray2[0];
        def desc = '-';
        strarray = str.split("<desc>");
        strarray2 = strarray[1].split("</desc>");
        desc = strarray2[0];
        def stock = '-';
        strarray = str.split("<stock>");
        strarray2 = strarray[1].split("</stock>");
        stock = strarray2[0];
        def deliv = '-';
        strarray = str.split("<deliv>");
        strarray2 = strarray[1].split("</deliv>");
        deliv = strarray2[0];
        def price = '-';
        strarray = str.split("<price>");
        strarray2 = strarray[1].split("</price>");
        price = strarray2[0];
        def discount = '-';
        strarray = str.split("<discou>");
        strarray2 = strarray[1].split("</discou>");
        discount = strarray2[0];
        def str2 = '<img src="/images/iPhone2.jpg"><br><hr><p>';
        str2 = str2 + '<table border="0" width="100%">';
        str2 = str2 + '<tr><td>Product Code</td><td style="text-align:right">' + codpro + "</td></tr>";
        str2 = str2 + '<tr><td>Description</td><td style="text-align:right">' + desc + "</td></tr>";
        str2 = str2 + '<tr><td>Unit Price</td><td style="text-align:right">' + price + " euro</td></tr>" ;
        str2 = str2 + '<tr><td>Qty Available</td><td style="text-align:right">' + stock + " pezzi</td></tr>";
        str2 = str2 + '<tr><td>Production time</td><td style="text-align:right">' + deliv + " gg</td></tr>";
        str2 = str2 + '<tr><td>Max Discount</td><td style="text-align:right">-' + discount + "%</td></tr>" ;
        str2 = str2 + "</table>";
```

A

B

The last modification is in the `ret` variable, where the stream of characters to be sent to the iPhone is stored. You will have to concatenate the string containing everything. So replace the instruction:

```
ret = ret + ". Data from dummy service was <b>" + str + "</b>";
```

with:

```
ret = ret + str2 + "<br><hr>";
```

This is shown in Example 4-5 in bold.

Example 4-5 Modification to accommodate the stream of characters

```
// ITSO
        ret = ret + str2 + "<br><hr>" ;
//       ret = ret + ". Data from dummy service was <b>" + str + "</b>";
```

4.3.3 Testing the application

Once all the modifications have been made to the code, it is possible to start the application and test whether it works correctly. To do that, click the green arrow on the CLEAR application row on the “My Applications - AppBuilder” page on the AppBuilder site:

`http://localhost:8070/, .`

Once started, it is possible to do a partial test directly on your Firefox Web browser by clicking the application link:

```
http://localhost:8080/
```

If you plan to access it with the Apple iPhone, use the IP address of the PC you are using with sMash, instead of localhost. You can call this page from the Safari browser inside Apple iPhone.

The second step will be to add an application link in iPhone as an icon. This can be done by pressing the **+** button on the browser page (you can find the + signal on the bottom navigator of the page, as shown in Figure 4-10).

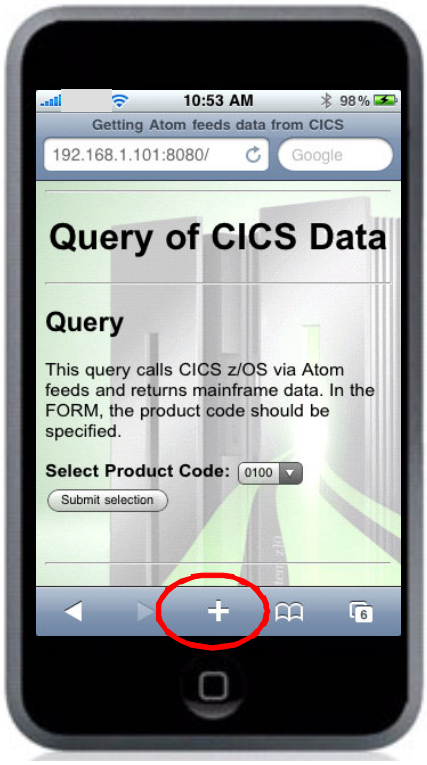


Figure 4-10 Adding the link to iPhone

After pressing the plus sign, a new window appears and you can specify a fast path to this application by selecting **Add to Home Screen**, as shown in Figure 4-11 on page 68.

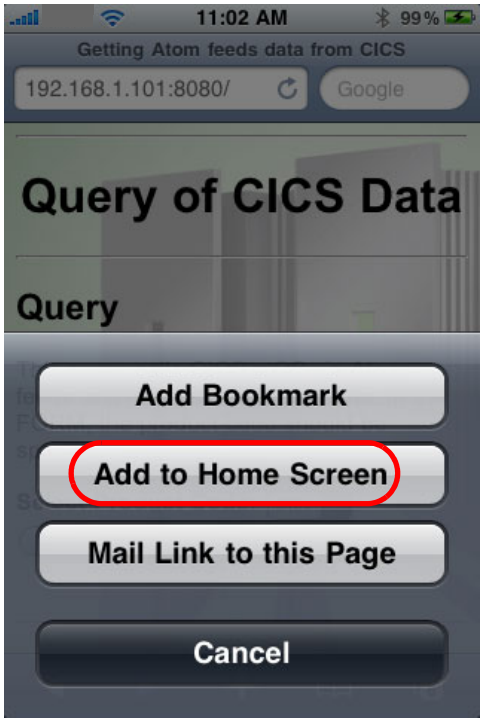


Figure 4-11 Add to Home Screen

You have the opportunity to change the name that appears for your icon on your iPhone (see Figure 4-12).

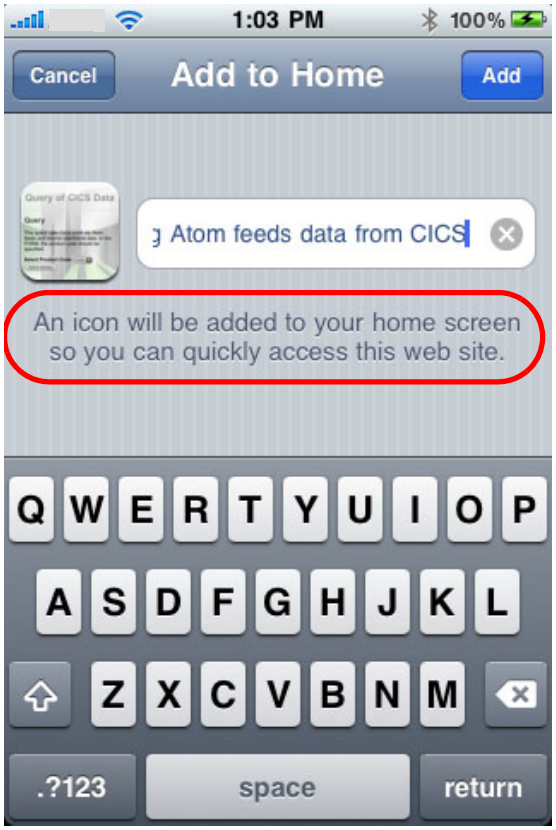


Figure 4-12 Adding the icon to the home screen

Once added, the icon related to this application is included on your iPhone home screen and every time you press the icon, you will execute your new application. As you can see from Figure 4-13, we named ours “Atom Sample”.

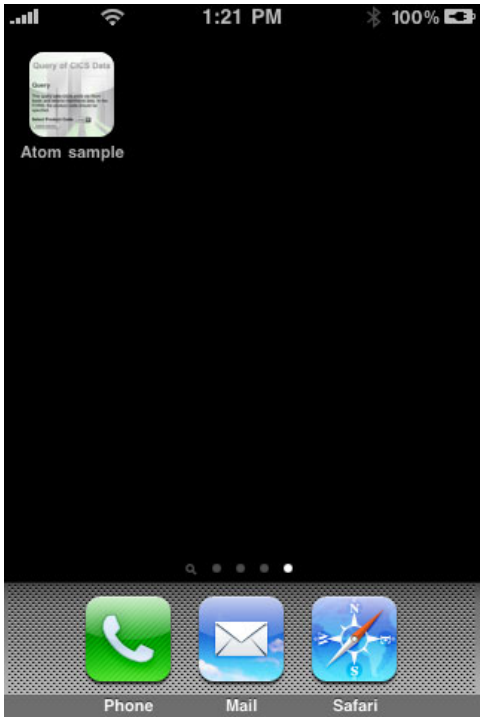


Figure 4-13 Application on home screen

When you click the icon, the application starts and an image with the word ”Loading” appears. This is done to reduce the perceived waiting time in case of network delays.

Figure 4-14 Loading the application

When the initial HTML page is loaded (Figure 4-15), the application now fills the screen because you no longer have a need for the plus sign because you have already added the application to your phone. It now has the appearance of a client application rather than a Web application.

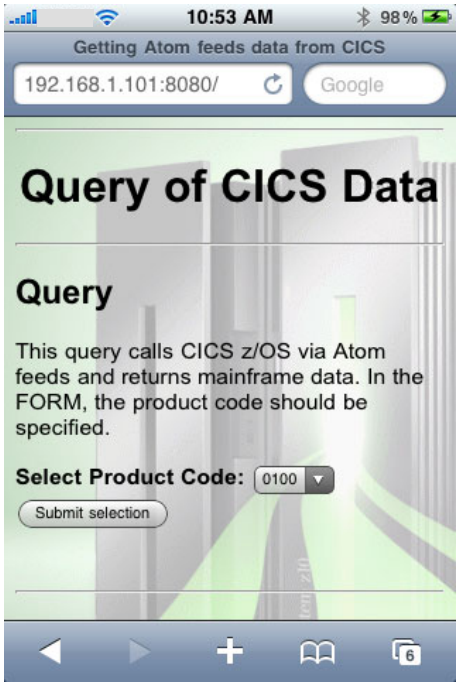


Figure 4-15 Application running

You can choose the product code numbers from a list (Figure 4-16). We chose to do that in order to test the list facility, but the form can be any kind of your choosing.

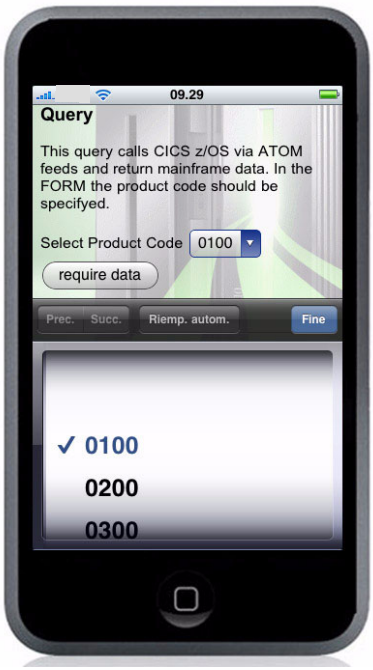


Figure 4-16 Selecting product codes

Clicking **Submit selection** submits the transaction request to CICS on z/OS. The request will be sent, the programs will manage the data, and an HTML page with the data will be returned to the Apple iPhone.

5

Accessing mainframe resources from Android mobile device applications

In this chapter we discuss approaches to accessing mainframe resources from mobile devices such as Android phones and Apple iPhone. In particular, the focus is on accessing mainframe resources from native applications on mobile devices, as opposed to accessing mainframe resources from a Web browser.

Our examples are developed for Android phones because the development tools are readily available for use. However, similar applications can be developed on other mobile devices such as Apple iPhone and devices running Windows Mobile.

Figure 5-1 on page 74 illustrates an overview of the two scenarios we describe in this chapter. The top part (above the line) illustrates the scenario described in 5.3, “Accessing mainframe resources from native smartphone applications” on page 77 in which the records stored in a VSAM file are made available via a REST service. The bottom part (below the line) illustrates the scenario described in 5.4, “Pushing information to smartphones” on page 83 in which we demonstrate a simple publish/subscribe model of communication.

© Copyright IBM Corp. 2010. All rights reserved.

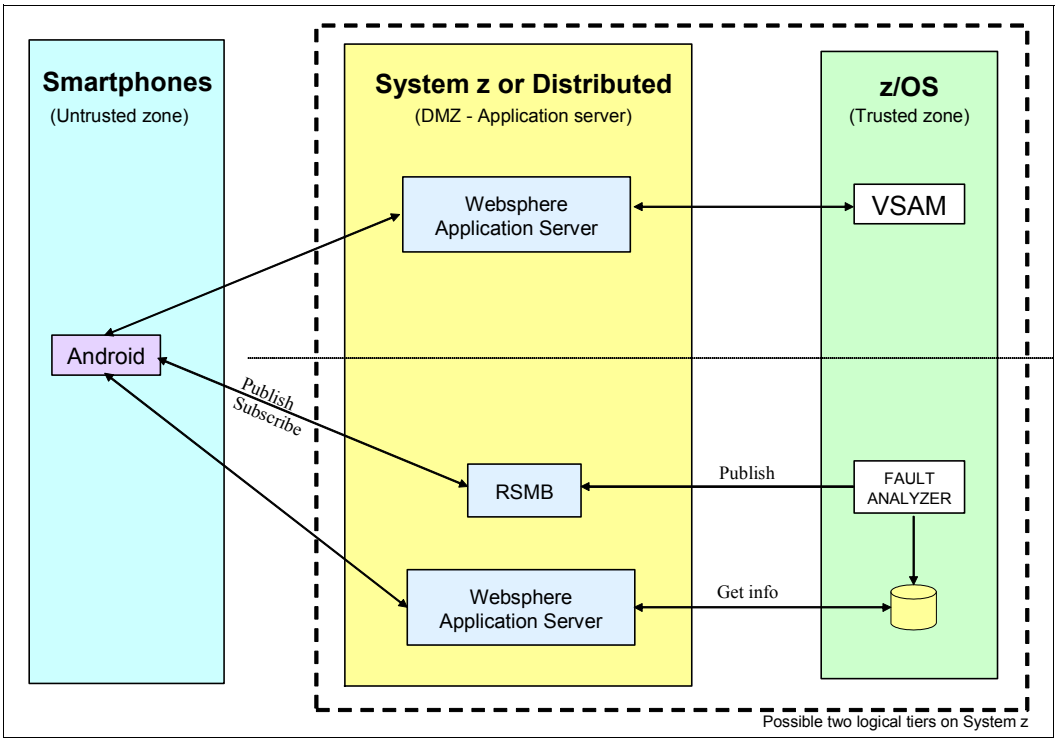


Figure 5-1 Solution architecture - Accessing z/OS applications from native Android applications

Sample code: All code for the scenarios discussed in this chapter is downloadable from the Web. Refer to Appendix D, “Additional material” on page 203 for details.

5.1 Setting up the development environment for Android applications

The Android Web site at:

http://www.android.com

provides comprehensive information about developing Android applications, including many podcasts and articles that describe Android architectures, development tools, and much more. In this section, we describe the development environment and tools we used for this book. However, we would recommend visiting Android.com for the latest information.

The following tools were used to develop the sample applications in this chapter:

- Eclipse 3.4.2, downloaded from:

  http://www.eclipse.org

- Android SDK 2.1, downloaded from:

  http://developer.android.com/sdk/index.html

- Android Development Tools.For instructions on how to install Android Development Toolkit for your Eclipse environment, visit:

  http://developer.android.com/sdk/eclipse-adt.html

- Sun JDK 6 Update 18, downloaded from:

  http://java.sun.com/javase/downloads/index.jsp[1]

http://developer.android.com/sdk/index.html describes the step-by-step instructions on how to set up your development environment. As an overview, you would perform the following steps:

1. Install the JDK and Eclipse.
2. Install the Android SDK.
3. Install the Android Development Tools plug-in for your Eclipse environment.
4. Configure the Android Development Tool by pointing it to the installation directory of the Android SDK.

Once the installation is complete, the Android Development Tools plug-in adds a number of tools to the Eclipse's Java perspective, as shown in Figure 5-2 on page 76.

To install additional samples or obtain information on installing samples, visit:

http://developer.android.com/resources/samples/get.html

5.2 Getting started with developing Android applications

Android SDK and Eclipse's Android Development Tools make the development of Android applications very simple. We found that the samples provided by the tools are the best way to get started, in particular ApiDemos.

1. Create a new Android project by selecting **File** → **New** → **Android Project**.
2. Give the project a name. As shown in Figure 5-2, we called this project "Api".

[1] At the time of writing, we were only able to use Sun's JDK to develop and test Android applications in the development environment described in this chapter.

3. Specify **Create project from existing sample**, and select **ApiDemos** (see Figure 5-2). In order to find these samples, you will have to select a target. That target is dependent on your version of the Software Developer Kit (SDK). At the time of writing, we were using an older version of Android (Version 1.1), so we found that the ApiDemos used a target of Android 1.1. The Android Open Source Project v2.1 has an API level of 7, so it uses the target named Android 2.1. You can find the sample sources on your computer in this location:

   `<sdk>/samples/<platform-version>/`

   On our system, we had `android-sdk-windows\samples\android-7` where `android-7` indicates our API is at level 7.

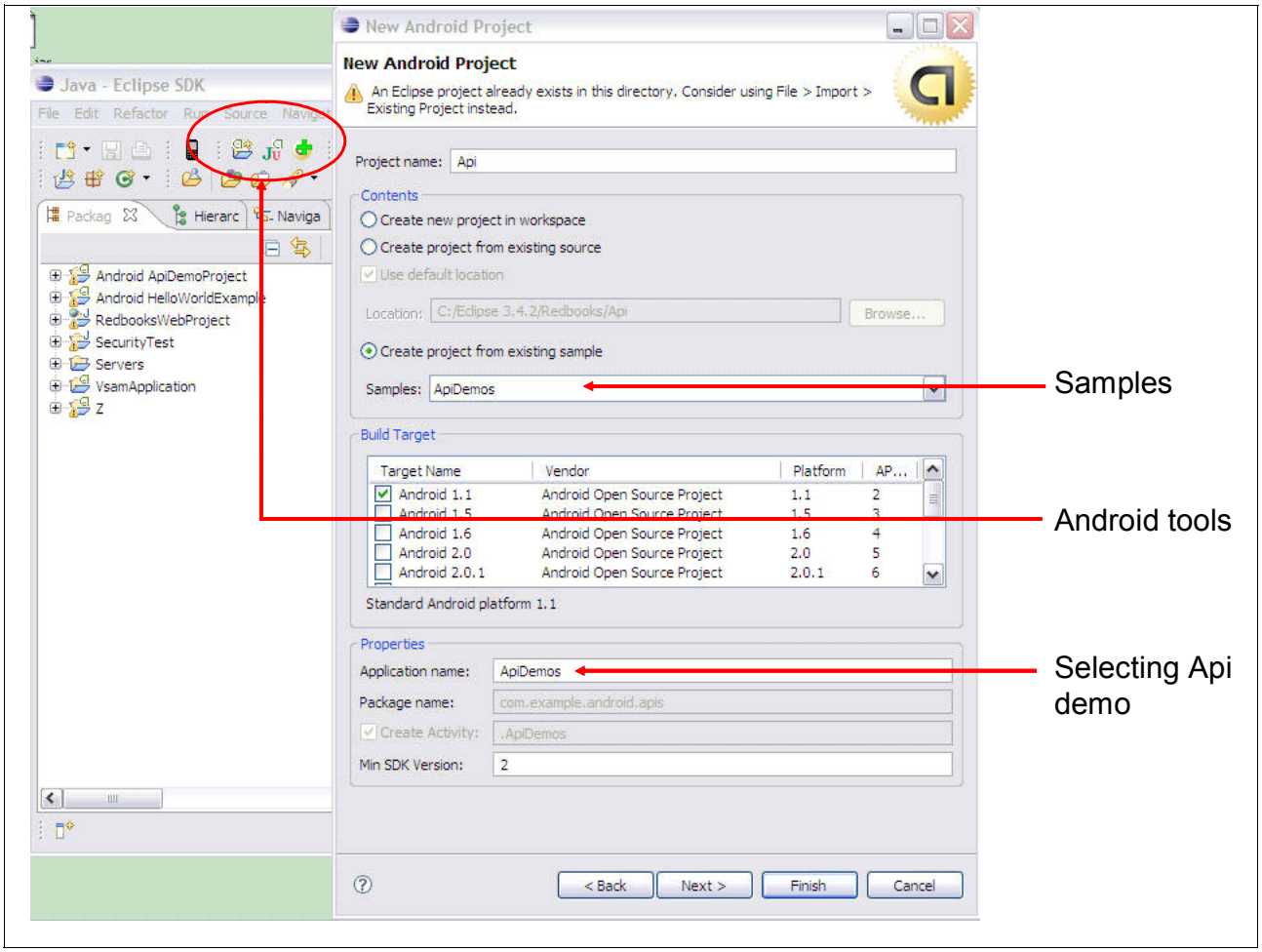


Figure 5-2 Java perspective with Android tools and creating Android sample project

4. Click **Finish**.
5. Run it for testing. Bring up the context menu against the project by right- clicking the “Api” project and select **Run** → **As Android Application**.
6. The sample Android application is deployed to the simulator, which is provided with the Android SDK, as shown in Figure 5-3 on page 77.

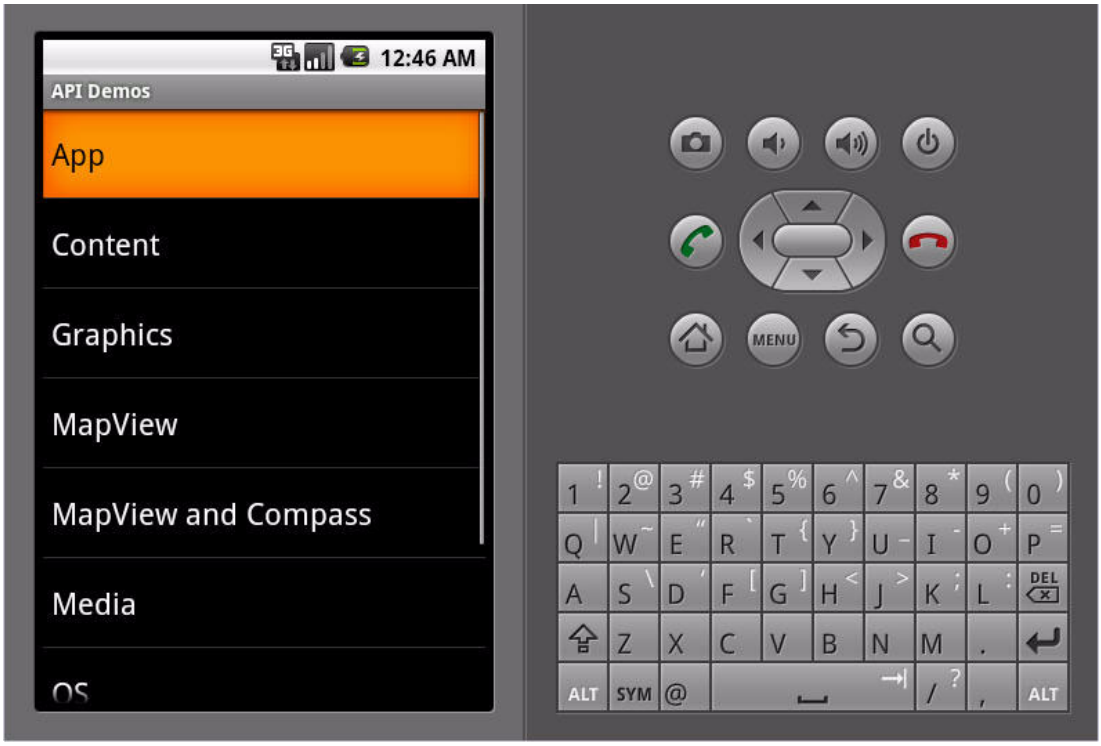


Figure 5-3 Android ApiDemos example in a simulator

5.3 Accessing mainframe resources from native smartphone applications

In this section, we demonstrate an approach to develop a simple Android application to access a VSAM file that resides on an IBM System z mainframe system.

5.3.1 Setting up the infrastructure

Figure 5-4 on page 78 provides an architectural overview of how this example is implemented, and Figure 5-5 on page 79 shows the user interface of this application.

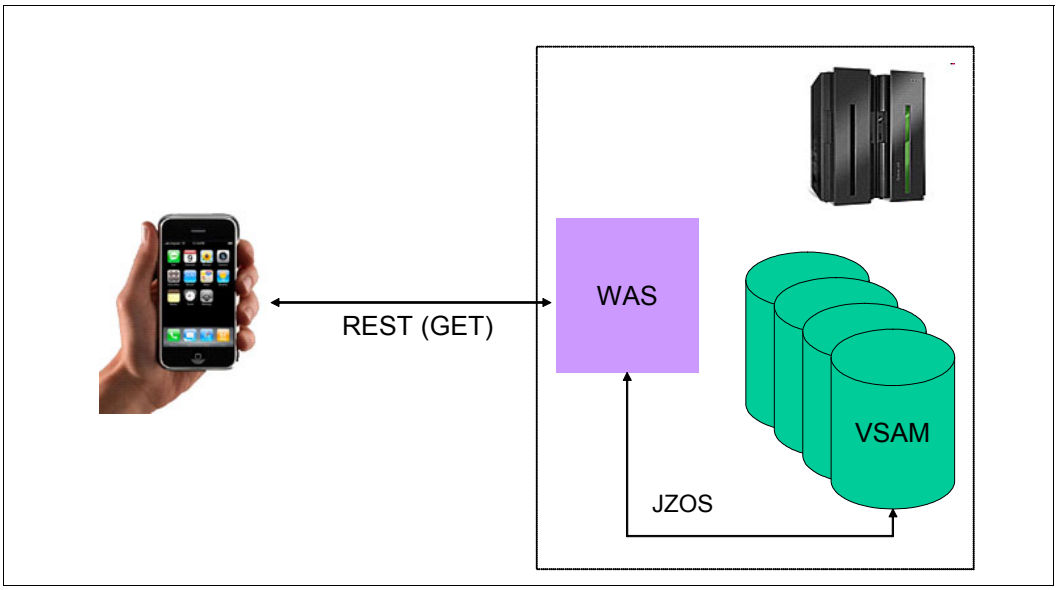


Figure 5-4 An overview of the VSAM access application

In this simple application, records are retrieved from a VSAM data set and displayed in an Android smartphone device. The application allows users to specify a number of records to retrieve from a VSAM file, and to scroll up and down records for display.

A native Android application was developed to implement the GUI on the devices. For a more detailed discussion on how to implement native applications for Android devices[2], refer to:

`http://developer.android.com/index.html`

A servlet which runs in a WebSphere Application Server environment implements a REST service to retrieve records from a VSAM file. The servlet handles HTTP GET requests with optional parameters that specify the starting record position in the VSAM file and a number of records to retrieve. For example, the following pattern specifies the starting position of 5, and reads 10 records:

`http://myhost/RedbooksWebProject/VsamReaderServlet?start=5&total=10`

The servlet returns the record information in text for display on the Android client. Depending on your preference and requirements, it is advisable to return the results in a structured format (for example, XML or JSON).

Note: The names that would be shown in Figure 5-5 on page 79 have been overlaid, for privacy reasons.

[2] The source code for the sample application is provided with this IBM Redbooks publication.

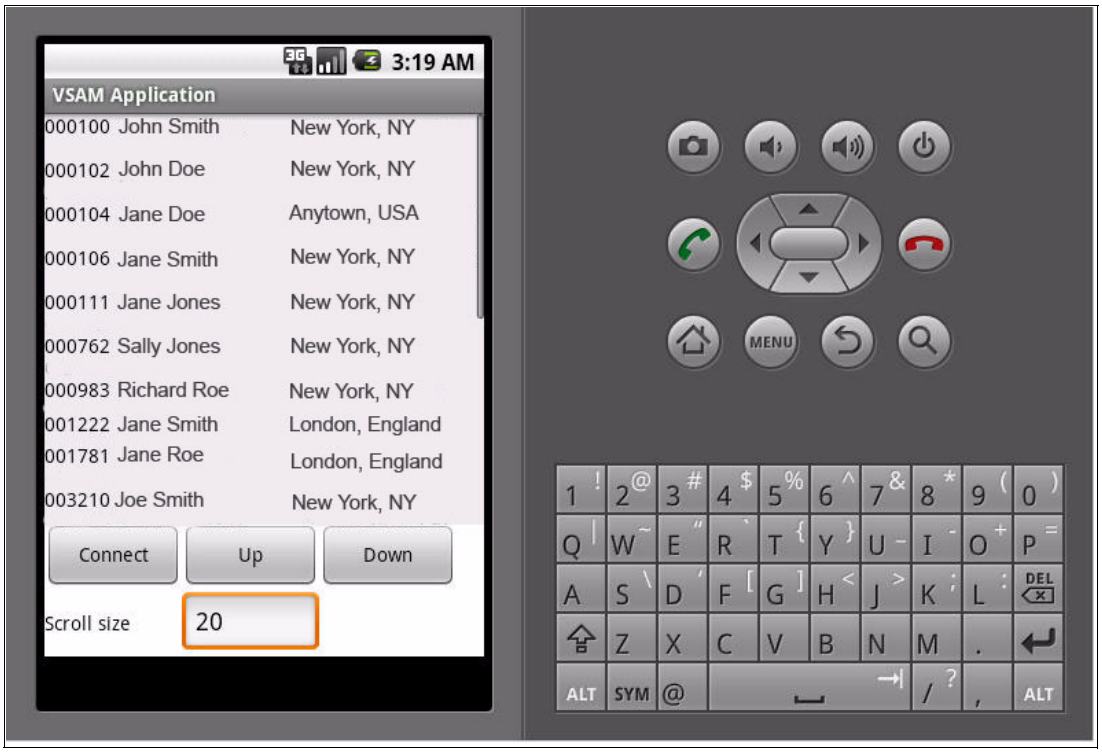


Figure 5-5 A simple application to access records stored in a VSAM file

In the sample Android application, it connects and retrieves the information using the standard Java APIs that are available in the Android runtime environment[3]. Example 5-1 shows sample code to access and retrieve information from a REST service. Refer to the `readData()` method in the `com.ibm.redbooks.android.vsam.VSAMReader.java` in the `VsamApplication` project for a complete listing of the program.

Example 5-1 Sample code for accessing URL from Android applications

```
URL url = new URL("http://myhost/SimpleVSAMApplication/VsamReaderServlet");
URLConnection connection = url.openConnection();
InputStream input = connection.getInputStream();
StringBuffer response = new StringBuffer();
int len = 0;

while ((len = input.read(buffer)) > 0) {
   response.append(new String(buffer, 0, len));
}
```

5.3.2 Steps for securing the simple application

The simple example provided in the previous section is unrealistic because we did not implement any security to secure the application. In this section, we describe an approach to securing the application.

[3] Android requires `android.permission.INTERNET` to be set for your application to access the Internet.

Securing the WebSphere component

We now describe an approach to securing the WebSphere component. However, you are encouraged to read other articles about this topic for a more in-depth discussion.

Setting up RACF to allow access to VSAM files

Servlets in a WebSphere environment typically run under the application server's generic user ID. You can either:

1. Define a RACF® profile so that the VSAM files are readable by the server's ID, or
2. Specify the *SynchToOSThread* option in your Web application so that the access to the VSAM files is controlled by the user's RACF settings.

Implementing authentication and secured communication in the WebSphere application

You can follow these steps to specify the security settings for your Web application[4]:

1. Open the Web Deployment Descriptor (web.xml) of your Web application (see Figure 5-6 on page 81).
2. Under the security tab, define a security role for your application (for example, VSAM Example Users).
3. Define a security constraint (for example, VSAM example user constraints), select all HTTP methods (GET, PUT, POST, and so on), and specify a URL pattern (for example, /* for everything).
4. For the newly defined constraint, select CONFIDENTIAL in the user data constraint. This option specifies the secured data transport using SSL.
5. Save your Web Deployment Descriptor.
6. Open the Application Deployment Descriptor (application.xml) and open the security tab (see Figure 5-7 on page 81).
7. Press **Gather** to pick up the security configuration from the Web application. You should see VSAM Example Users listed in the view.
8. Select **VSAM Example Users**, and tick the **All authenticated users** box under the WebSphere Bindings option.

When these steps are followed correctly, the Web application is only accessible to those users who are able to authenticate with the server (RACF by default on z/OS), and data is transported securely using SSL.

[4] We used Rational Developer for System V7.6.

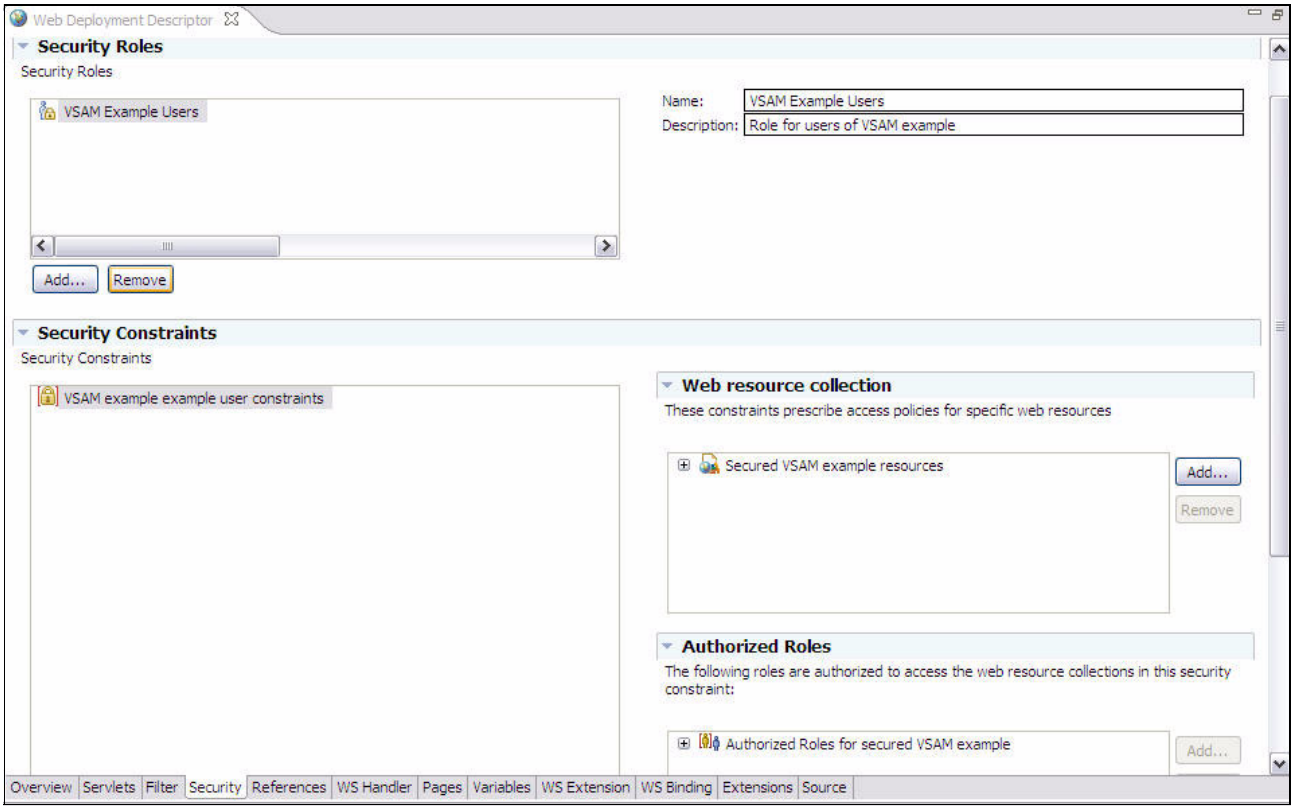


Figure 5-6 Security settings in web.xml

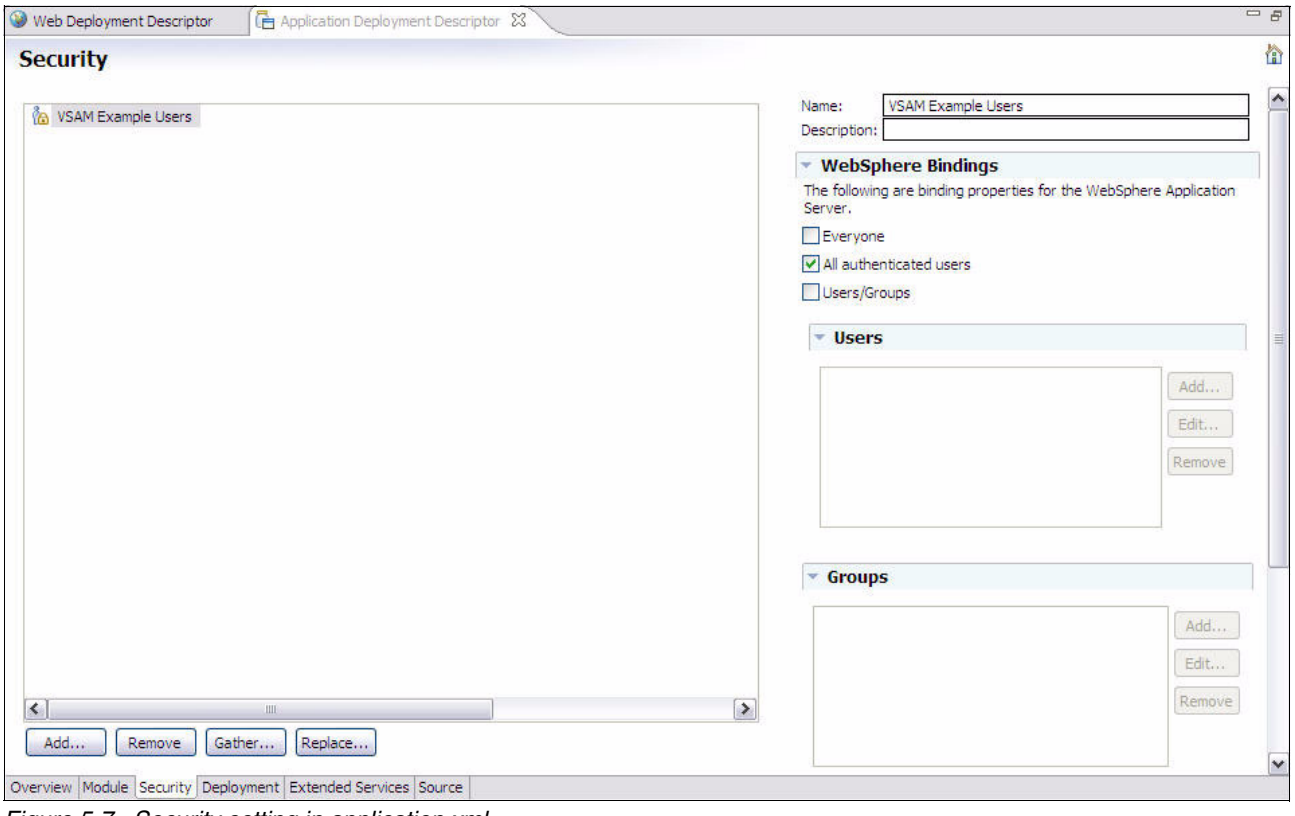


Figure 5-7 Security setting in application.xml

Implementing authentication and secured communication in Android

The simple Android application described in the previous section would not work now because of the security infrastructure in place. We performed the following two steps to secure the Android application:

1. Added an additional set of calls to the Java security classes to handle the basic authentication and secured communication. Again, the standard set of Java security APIs is available in the Android runtime environment (see Example 5-2)[5]. Refer to the `readDataSecured()` method in the `VSAMReader.java` class in the`VsamApplication` project.
2. Added GUI components to allow the entry of user name and password to meet the requirements of simple authentication (see Figure 5-8 on page 83).

Example 5-2 Sample code for managing authentication and secured communication

```
HostnameVerifier myHostNameVerifier = new HostnameVerifier() {
   public boolean verify(String hostname, SSLSession session) {
      if (hostname.equalsIgnoreCase("myhost")) {
         return true; // Only allow connection to my own host.
      } else {
         return false;
      }
   }};

X509TrustManager myManager = new X509TrustManager(){
   public void checkClientTrusted(X509Certificate[] chain, String authType)
         throws CertificateException {
   }
   public void checkServerTrusted(X509Certificate[] chain, String authType)
         throws CertificateException {
   }
   public X509Certificate[] getAcceptedIssuers() {
         return new X509Certificate[0];
   }};

HttpsURLConnection.setDefaultHostnameVerifier(myHostNameVerifier);
SSLContext context = SSLContext.getInstance("TLS");
TrustManager managers[] = {myManager};
context.init(null, managers, new SecureRandom());

HttpsURLConnection.setDefaultSSLSocketFactory(context.getSocketFactory());

URL url =
   new URL("https://myhost/SecuredSimpleVSAMApplication/VsamReaderServlet");
HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();

byte[] result = Base64.encodeBase64((userID + ":" + password).getBytes(), true);
String str = StringUtils.newStringUtf8(result).trim();
connection.setRequestProperty ("Authorization", "Basic " + str);

InputStream input = connection.getInputStream();

...
```

[5] Base64 encoder is not available as part of Android SDK and runtime environment. We used Apache's commons codec library.

Note: The names that would be shown in Figure 5-8 have been overlaid for privacy reasons.

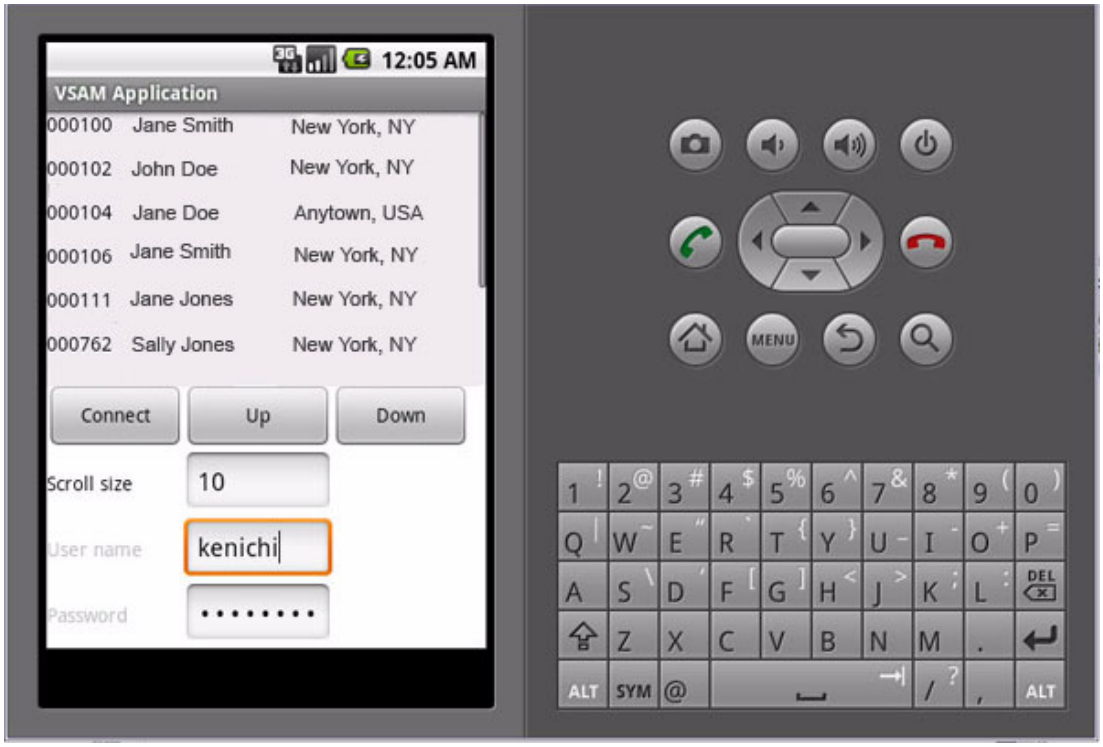


Figure 5-8 VSAM example with user name and password entry fields

5.3.3 Extending the architecture to access more mainframe resources

The simple example provided in this section can be extended further to access a variety of mainframe resources including DB2, CICS, IMS and much more. At the time of writing, there is no native SOAP support in Android SDK[6]. For this reason, it appears to be the easiest approach to make your resources available via REST services using the approach described in this section.

For example, we can extend the servlet to access DB2 databases using a Java Database Connectivity (JDBC) connection, accessing CICS and IMS using Web Services and Atom feeds, and invoking native applications via JNI calls, just to name a few possibilities.

Once the mainframe resources are made available securely as described in this chapter, they are available for any mobile devices such as Android devices, iPhones, and other smart devices.

5.4 Pushing information to smartphones

The scenario we described in the previous section requires users to proactively access mainframe systems to retrieve information. In this section, we demonstrate another paradigm,

[6] You can always download additional libraries to implement. For example, kSOAP (http://ksoap2.sourceforge.net/).

publish/subscribe. It is an asynchronous messaging paradigm where senders (publishers) of messages are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into classes (categories), without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

The approach offers a number of advantages:

- Allows mainframes to publish information to subscribers to draw their attention when needed (for example, job completion and failure).
- A more friendly approach for the battery consumption of mobile devices (continuous wireless access typically leads to poor battery performance).
- Heavy-lifting of information gathering is left on the server side (for example, an agent to monitor the disk space availability).

In this section, we describe an approach based on the MQ Telemetry Transport (MQTT) protocol, which enables a publish/subscribe messaging model in an extremely lightweight way. This protocol is particularly useful when a small code footprint is required and/or network bandwidth is at a premium[7]. Depending on the requirements of a scenario, you may want to consider other solutions based on other middleware products such as WebSphere MQ and Message Broker, WebSphere MQ Everyplace®, and Lotus® Expetidor just to name a few.

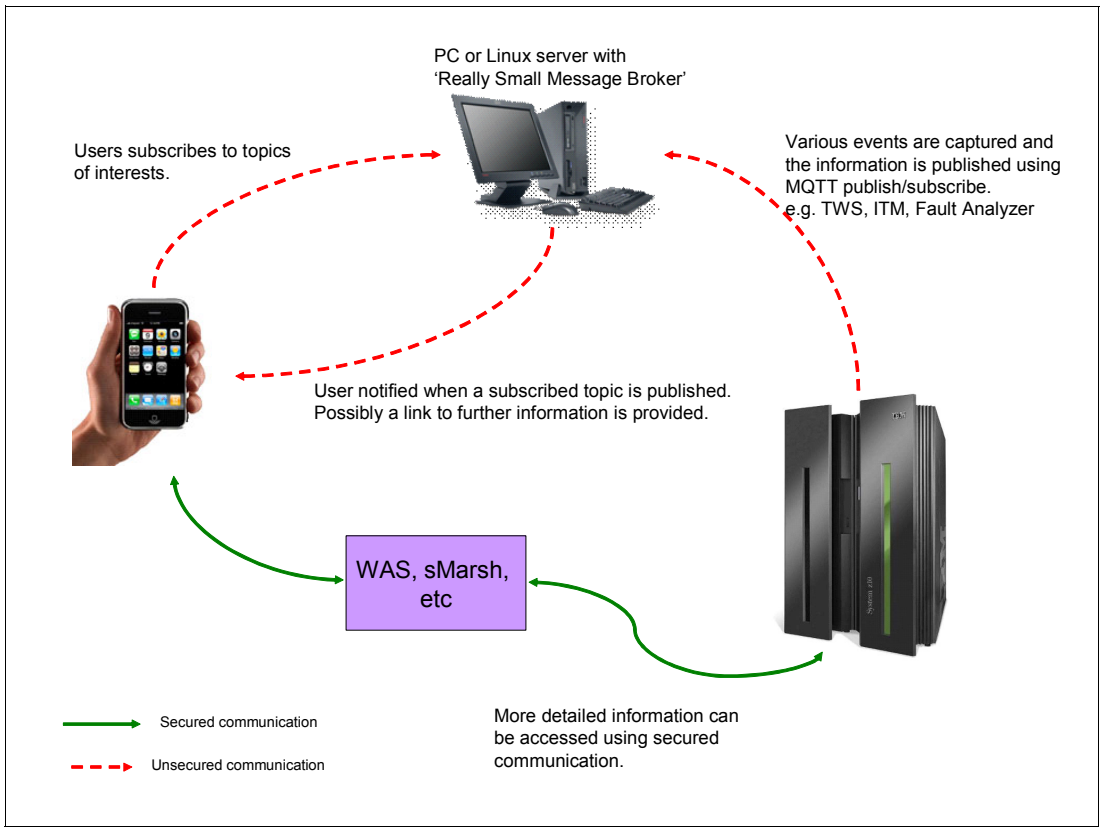


Figure 5-9 Architectural overview of the publish/subscribe scenario

Figure 5-9 illustrates an architectural overview of the publish/subscribe scenario. There are secured (solid green) and unsecured (dashed red) communication channels in the figure. The publish/subscribe based on the MQTT protocol is implemented based on an unsecured

[7] http://mqtt.org/

communication channel[8]. This approach is arguably acceptable if we perceive the purpose of publish/subscribe solely as a way to notify subscribers of an event on the server—hence it should not contain any confidential information. The actual data (that is, confidential information) should be transmitted to subscribers using secured solutions (such as an example from the previous section). In our example below, the publish/subscribe message contains a short description of an event. Upon receiving a notification, the user is provided with an option to access further information via a secured communication channel.

5.4.1 Scenario description: notification of abnormal job termination

Fault Analyzer for z/OS[9] is a member of the Problem Determination Tools Family, which assists developers in analyzing and fixing application and system failures. A typical usage of this product is as follows:

> A job/application fails and terminates abnormally (abend) in your production system. Fault Analyzer for z/OS is invoked to analyze the situation and provides the diagnostic information.

At this point, an application developer typically examines the diagnostic information and determines whether any action is required to recover from the failure. The recovery action may include modifications to the application, configuration to the run-time environment, or allocation of additional resources, just to name a few possibilities.

The publish/subscribe approach fits in well with this scenario because it notifies the interested parties (for example, application developers and system programmers) as soon as a failure occurs on the system.

Figure 5-10 on page 86 illustrates an overview of how this is implemented.

[8] The MQTT protocol does not support secured communication. However, it is possible to encrypt the data if necessary.

[9] http://www-01.ibm.com/software/awdtools/faultanalyzer/

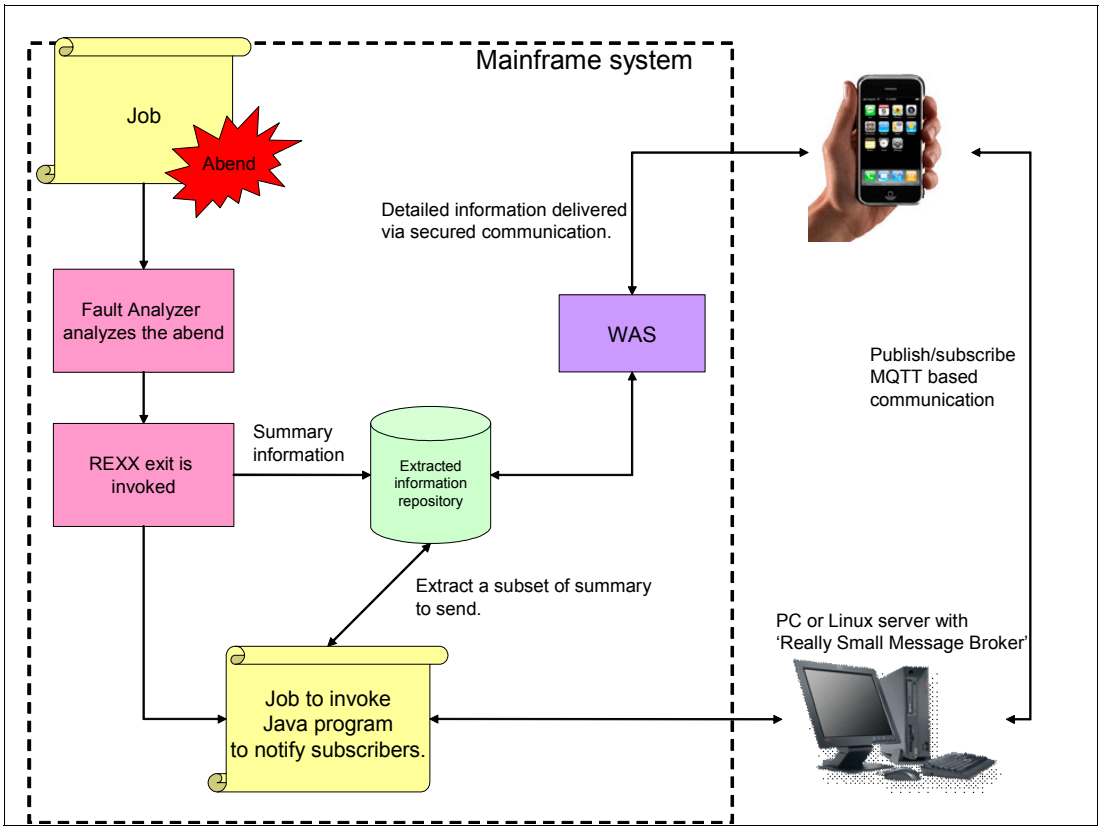


Figure 5-10 An overview of the Fault Analyzer publish/subscribe scenario

From the publisher's perspective (mainframe production environment with Fault Analyzer):

1. An application terminates with an unexpected failure condition.
2. Fault Analyzer is invoked to analyze the environment to provide diagnostic information.
3. Fault Analyzer provides a set of user exits to allow custom actions. For the purpose of the publish/subscribe scenario, a REXX user exit is used to extract the subset of diagnostic information such as its job name, abend code, and date and time.
4. The extracted information is stored in a data set.
5. The REXX exec invokes a Java program to publish the subset of extracted diagnostic information.
6. The MQTT server notifies the subscribers.

From the subscriber's perspective (Android client):

1. Connects to a server (MQTT server).
2. Subscribes to interested topics.
3. The subscriber is notified when a new message is published for a subscribed topic.
4. In our implementation of the Android client, the user is offered the possibility to retrieve more detailed information using the secured communication.
5. If the user selects to retrieve further information, more detailed diagnostic information is downloaded via WebSphere Application Server (the same approach as in 5.3.2, "Steps for securing the simple application" on page 79).
6. The detailed diagnostic information is displayed on the client. Otherwise, no action is taken.

5.4.2 Software for supporting publish/subscribe applications

A list of software products used to support the scenario is as follows:

The MQTT protocol is an open standard. IBM provides a client and a server implementation.

- MQTT Java client at:

  http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006006

- IBM Really Small Message Broker at:

  http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006006

The following software needs to be installed and configured on the server:

- WebSphere Application Server V7
- Fault Analyzer for z/OS V10

The sample Android application was developed using the Android's SDK in Eclipse (see 5.2, "Getting started with developing Android applications" on page 75).

5.4.3 Steps for developing publish/subscribe applications

In this section, we describe the steps for developing a publish/subscribe application.

Step 1 - Consideration for defining topics for publish/subscribe

Defining well-structured topics for publish/subscribe applications is an important step to ensure that subscribers receive the right set of published messages.

The MQTT protocol (and many others) allows application developers to define publication topics in a hierarchical manner. By having a hierarchical structure, it enables subscribers to adjust the level of notification to receive—from subscribing to a very specific topic to a more generic topic that includes many subtopics. A sample topic hierarchy is:

```
/news
/news/business
/news/business/retail
/news/business/manufacturing
/news/business/finance
/news/sports
/news/sports/baseball
/news/sports/golf
/news/sport/soccer
```

The most generic topic is `/news`. If a user subscribes at this level, the user is notified whenever a new news is posted to the server. On the other hand, if you are only interested in a very specific news (for example, `/news/sports/baseball`), then the model allows the user to subscribe to that very specific news.

In our example, we defined topics as follows:

```
/jobInformation
/jobInformation/abendInfo (for supporting Fault Analyzer scenario)
/jobInformation/TWS (for supporting event from TWS).
```

The MQTT programming APIs allow the specification of topics using operators such as # and +. # matches every topic at any number of levels, whereas + only represents a level in a hierarchy.

The topic hierarchy and its operators, provide the capability for subscription by people in different roles in a single solution framework. For example, application developers are only interested in situations where their applications terminate abnormally. So they would subscribe to the `/jobInformation/abendInfo` topic. On the other hand, system programmers on duty would subscribe to every job information by specifying `/jobInformation/#`.

Step 2 - Setting up a server for publish/subscribe applications

We used Really Small Message Broker (RSMB), which runs on a variety of platforms including the Linux on IBM System z platform. We used a Windows version (which was downloadable with a zipped file containing an executable program), and accepted the default settings.

Step 3 - Developing publisher and subscriber programs

We used IBM Java MQTT APIs to develop the publisher (mainframe component) and subscriber (Android application) applications. Connell and Stanford-Clark[10]explain the steps in detail. A summary is provided here.

The steps for publishers and subscribers are almost identical. The following two lines establish a connection to a MQTT server. The first parameter of the `createMqttClient()` method specifies the IP address and port number of the server. The second line connects to the server. The first parameter specifies a name to uniquely identify the client. A unique name must be specified to support multiple publishers and subscribers.

```
IMqttClient client = MqttClient.createMqttClient("tcp://address:1883", null);
client.connect("Client name", true, (short) 5);
```

The following line allows a client to publish a message to a topic.

```
client.publish("/jobInformation/abendInfo", message, 0, false);
```

The following line allows a client to subscribe to topics (the parameter type is `String[]`), so multiple topics can be subscribed simultaneously.

```
mqttClient.subscribe(topics, QoS);
```

For a client to receive a notification for subscribed topics, it must register an implementation of the `com.ibm.mqtt.MqttSimpleCallback` interface class.

```
client.registerSimpleHandler(new MqttSimpleCallback(){
  public void connectionLost() throws Exception {
    System.out.println("Connecton is lost!");
  }

  public void publishArrived(String topic, byte[] data, int QoS,
    boolean retained) throws Exception {
    System.out.println("New message arrived for " + topic);
  }
});
```

The following calls are required to disconnect from the server. The first call notifies the server that the client is ready to disconnect. The second call terminates all started processes (for example, listener).

```
client.disconnect();
client.terminate();
```

[10] http://www.ibm.com/developerworks/websphere/library/techarticles/0508_oconnell/0508_oconnell.html

Step 4 - Setting up the infrastructure to provide more information

In our sample application, a user is offered to retrieve additional diagnostic information about the failed job from the server. Such action typically requires an additional level of security so that only authorized subscribers (for example, only users with a valid user name and password with valid RACF access to the resources) are allowed to retrieve the information.

In our example, we used the same approach we described in 5.3.2, “Steps for securing the simple application” on page 79.

Step 5 - Preparing a subscriber application to publish messages

In our example, invocation of Fault Analyzer is a trigger point to publish a message to the `/jobInformation/abendInfo` topic. So we developed a REXX exec to invoke the publisher Java code (from Step 3 - Developing publisher and subscriber programs) to publish the information.

The approach in this step is largely decided by your trigger point (what constitutes an event worthy of publishing messages) and the availability of a technical solution to invoke MQTT client calls.

In our example, a REXX exec is provided to the Fault Analyzer user exit to publish the message. We invoke the Java program from our REXX exec by submitting a job, which invokes the Java program by using JZOS[11].

Step 6 - Running the application

Figure 5-11 on page 89, Figure 5-12 on page 90, and Figure 5-13 on page 90 are the set of panel shots from the sample Android application, which illustrate the flow of events and user interactions.

Figure 5-11 Main panel for the sample Android application

[11] http://www-03.ibm.com/servers/eserver/zseries/software/java/products/jzos/overview.html

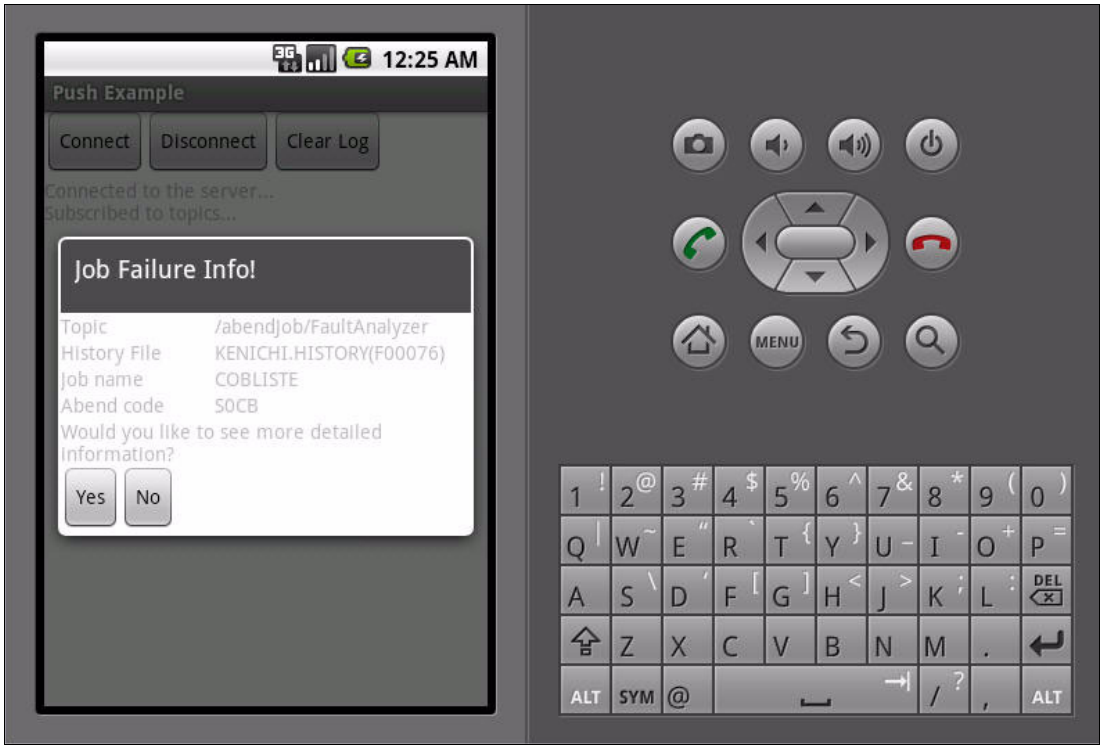


Figure 5-12 Notification from the publisher

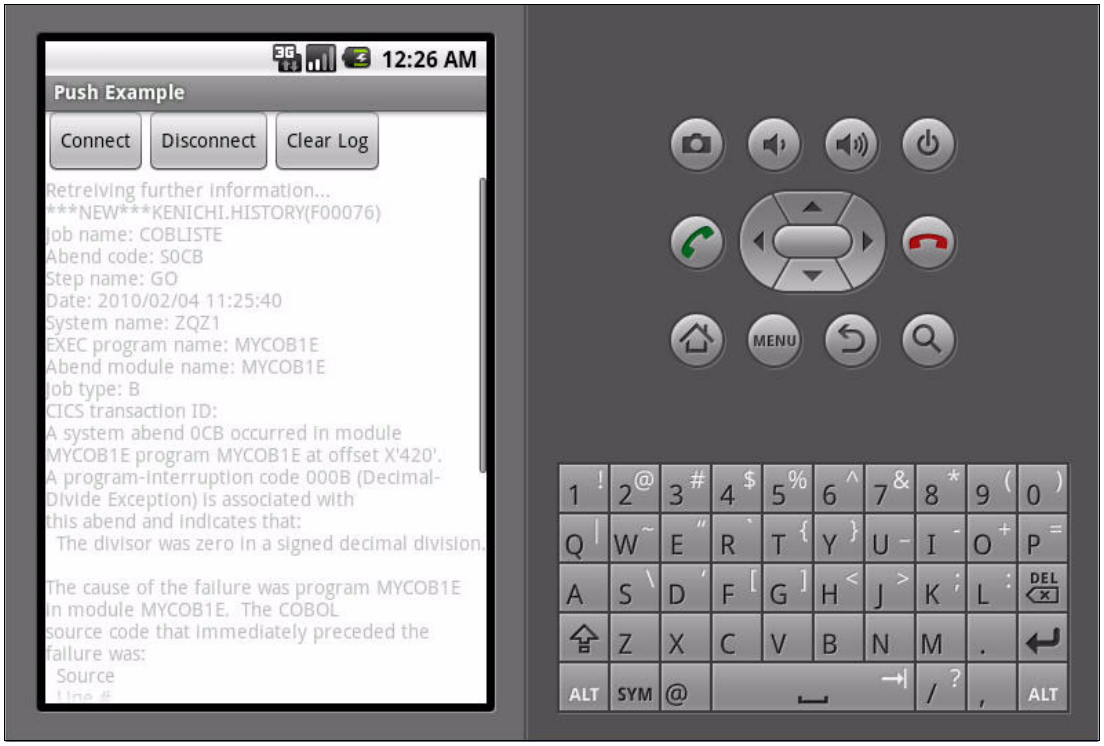


Figure 5-13 Display of detailed diagnostic information

5.4.4 Extending the sample application

The sample application we described in the previous sections can be extended to cover many scenarios. That is, we can install an exit to a variety of triggering points to publish messages to subscribers. Some possibilities are:

- Adding an additional step at the end of a job (JCL) to publish the highest RC from the steps in the job. A sample Java program is included here (Example 5-3); it can be invoked from your job via JZOS job launcher.
- Including the notification step by extending the Tivoli Workload Scheduler (TWS)[12] exit points.
- Extending your applications to include calls to publish messages.

Example 5-3 Sample Java client to publish messages

```
import com.ibm.mqtt.*;

public class MqttPublishClient {
  public static void main(String[] args) {
    try {
      System.out.println("Starting now...");

      if (args.length != 3) {
        System.err.println(
          "ERROR: MqttPublishClient [ip address] [topic] [message]");
        return;
      }
      String ipAddress = args[0];
      String topic = args[1];
      String msg = args[2];

      System.out.println("Connecting to " + ipAddress + "...");
      System.out.println("topic: " + topic);
      System.out.println("message: " + msg);

      IMqttClient client = MqttClient.createMqttClient(
        "tcp://" + ipAddress + ":1883", null);

      client.connect("MS" + System.currentTimeMillis(), true, (short) 5);

      client.publish(topic, msg.getBytes("US-ASCII"), 0, false);

      client.disconnect();
      client.terminate();

      System.out.println("Done!");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

[12] http://www-01.ibm.com/software/tivoli/products/scheduler/

6

Accessing a CICS Web service with an EGL Rich UI client on Apple iPhone

In this scenario we discuss how to extend a CICS Web service to an Apple iPhone application with the help of Enterprise Generation Language (EGL) Rich User Interface (UI).

© Copyright IBM Corp. 2010. All rights reserved.

6.1 High-level architecture

Figure 6-1 shows the high-level architecture of this scenario.

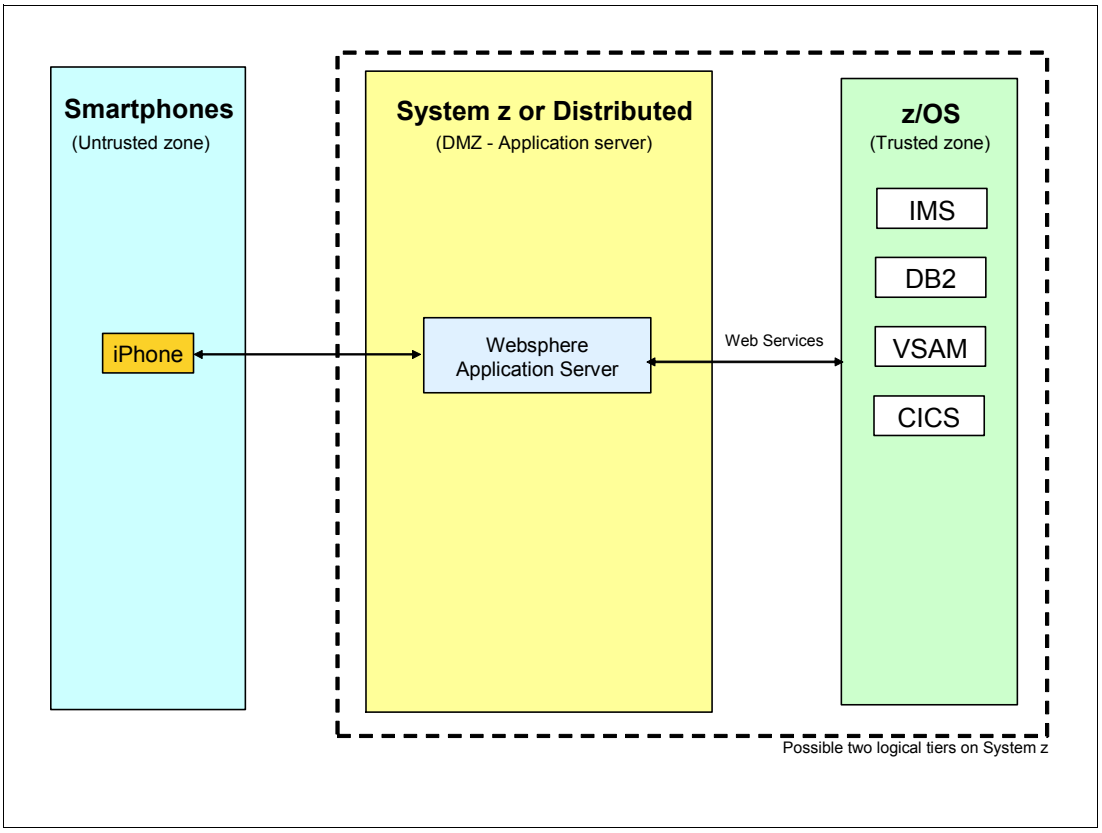


Figure 6-1 Solution architecture - Accessing z/OS applications using EGL Rich UI clients

The objective of this scenario is to access back-end data through a CICS/COBOL application running on z/OS, but using an iPhone as client. In this scenario the actual CICS/COBOL application will not change, and we use Enterprise Generation Language (EGL) Rich UI to develop a new smartphone user interface for the application.

6.2 Software used

We primarily used two software components to build, test and deploy this solution scenario:

- Rational Developer for System z (RDz) with EGL Version 7.6 or Rational Business Developer (RBD) Version 7.5
- WebSphere Application Server 7.0

You can use either RDz with EGL or RBD stand-alone, as both of these products provide EGL language support.

In our scenario we use EGL Rich UI capabilities for calling a CICS Web service and displaying the results returned on an Apple iPhone panel.

6.2.1 Enterprise Generation Language (EGL)

The IBM Enterprise Generation Language (EGL) is a high-level language that provides an additional layer of abstraction from lower-level programming languages and architectures. Specifically, EGL is a language that is not tied to a specific technology. It provides platform neutrality allowing a deployed application to be generated for a number of target environments. Developers are not required to learn multiple languages, but instead use the neutral language and transform it into the language that best suites the selected target environment (COBOL, Java). Code generation predictively reduces the cost and time needed to become proficient in designing and implementing applications in multiple languages and environments.

6.2.2 Rational Developer for System z with EGL

Rational Developer for System z (RDz) consists of a common workbench and an integrated set of tools that offers application development and maintenance, run-time testing, and rapid development and deployment of simple and complex applications. It offers an integrated development environment (IDE) with advanced, easy-to-use tools and features to support application development for multiple runtimes such as CICS, WebSphere, IMS and DB2. It helps developers rapidly design, code, and deploy complex applications.

RDz with EGL provides support for EGL, Cobol, Assembler, PL/I, C/C++, SQL, and stored procedures.

6.2.3 Rational Business Developer

Rational Business Developer (RBD) is an integrated development environment for Enterprise Generation Language (EGL) that can help a developer with developing and testing of EGL applications. RBD is an Eclipse-based IDE and can be integrated with WebSphere Application Server or Apache Tomcat for application debugging and testing.

For more information about Rational Business Developer, visit:

http://www-01.ibm.com/software/awdtools/developer/business/

6.2.4 WebSphere Application Server

WebSphere Application Server is the IBM runtime environment for Java-based applications. The application server acts as middleware between back-end systems and clients. It provides a programming model, an infrastructure framework, and a set of standards for a consistently designed link between them.

WebSphere Application Server is available on a wide range of platforms and in multiple packages to meet specific business needs. WebSphere Application Server Version 7.0 provides the runtime environment for applications that conform to the J2EE 1.2, 1.3, 1.4, and Java EE 5 (formerly J2EE 1.5) specifications.

For more information on Websphere Application Server, visit:
http://www-01.ibm.com/software/webservers/appserv/was/

6.3 Back-end requirements

In order to use this scenario with one of your CICS back-end applications you need to have a version of CICS supporting Web Services. Also, you need to have CICS Web Services configured and ready to use. Finally, you need to have the CICS application you wish to use enabled as a Web service. The WSDL file belonging to this CICS Web service is required in the development and testing process of the new smartphone EGL client UI.

Creating and deploying a Web service for CICS is not in the scope of this book and has been documented already in numerous other publications. A good starting point is the following IBM Redbooks publication:

http://www.redbooks.ibm.com/abstracts/sg247126.html?Open

6.4 EGL Rich UI client

We now discuss how to extend a CICS Web service to iPhone with the help of EGL Rich UI.

6.4.1 EGL Rich UI

EGL Rich UI provides you with the ability to generate JavaScript that runs in the browser. This is important because it makes the Web page more responsive, and provides greater flexibility so that the user experience can go beyond just submitting and receiving a page. For example, after the user clicks a radio button, the logic might respond by changing the contents of a text box. The change occurs quickly because the JavaScript runs locally and, in most cases, only needs to redraw one area of the page instead of the whole page.

An extension of client-side JavaScript is AJAX (Asynchronous JavaScript and XML), a technology that enables the run-time invocation of remote code and the subsequent update of a portion of a Web page, even as the user continues working elsewhere on the page. After the user selects a purchase order from a list box, for example, the JavaScript logic might request transmission of order-item details from the remote Web server and then place those details in a table displayed to the user. In this way, the application can access content from the server, but it saves time by selecting—at run time—which content is transmitted.

You can write Rich UI applications that use this technology using EGL syntax. We now explain, using Rational Developer with EGL, how you can create a Rich UI EGL project, import the WSDL to be used, create the Rich UI interface, and create the EGL logic to invoke the SOAP Web service.

6.4.2 Step-by-step instructions to create an EGL Rich UI Client application

Sample code

The code developed for this scenario is available for download as a Project Interchange File. Refer to Appendix D, “Additional material” on page 203 for instructions.

Getting started

Let us get started with the creation of a new EGL Project, as follows:

1. Open Rational Business Developer by selecting **Start** → **Programs** → **Rational Software Delivery Platform** → **Rational Business Developer** → **Rational Business Developer.** It will open up with the Workspace Launcher (Figure 6-2).

The workspace is a folder location on your local system where all the artifacts related to the project are stored. Specify the location of the folder in the Workspace field.

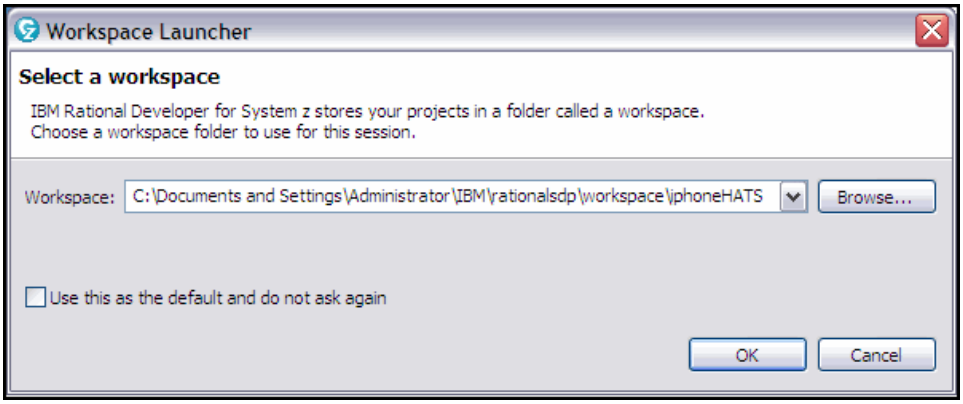


Figure 6-2 Specifying a folder location to store all project artifacts

2. Click **OK.**

   You will now see the Rational Business Developer workbench. And to work on an EGL Rich UI project, you need to switch to the EGL Rich UI perspective. You can do so by selecting **Window** → **Open perspective** → **Other** from the workbench menu bar and selecting **EGL Rich UI** from the pop-up window, as shown in Figure 6-3.

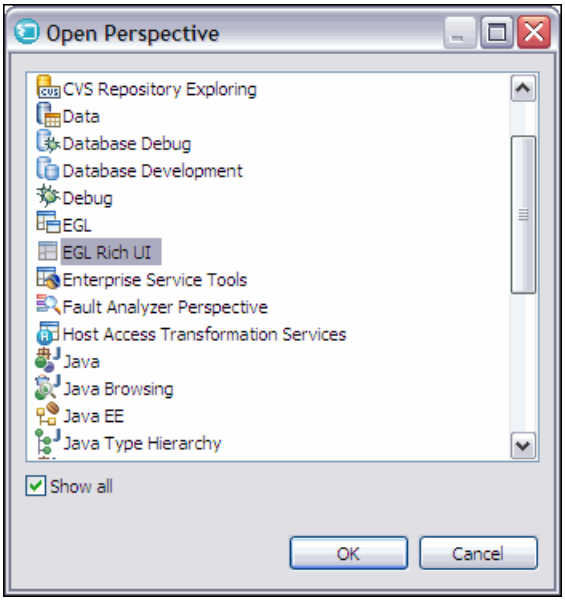


Figure 6-3 Selecting the EGL Rich UI perspective in RBD

3. Click **OK**.

   Once you are in the EGL Rich UI perspective, it is time to create a new EGL Rich UI project. To create a new project, select **File** → **New** → **EGL Project** from the workbench's menu bar, as shown in Figure 6-4.

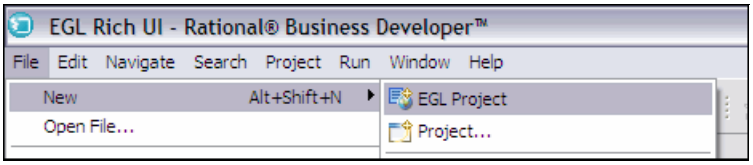


Figure 6-4 Creating a new EGL project

This opens a pop-up window for more details. You need to specify a project name and EGL project type, so let us insert the following details (Figure 6-5):

Project name iPhoneEGL

EGL Project Type Rich UI Project

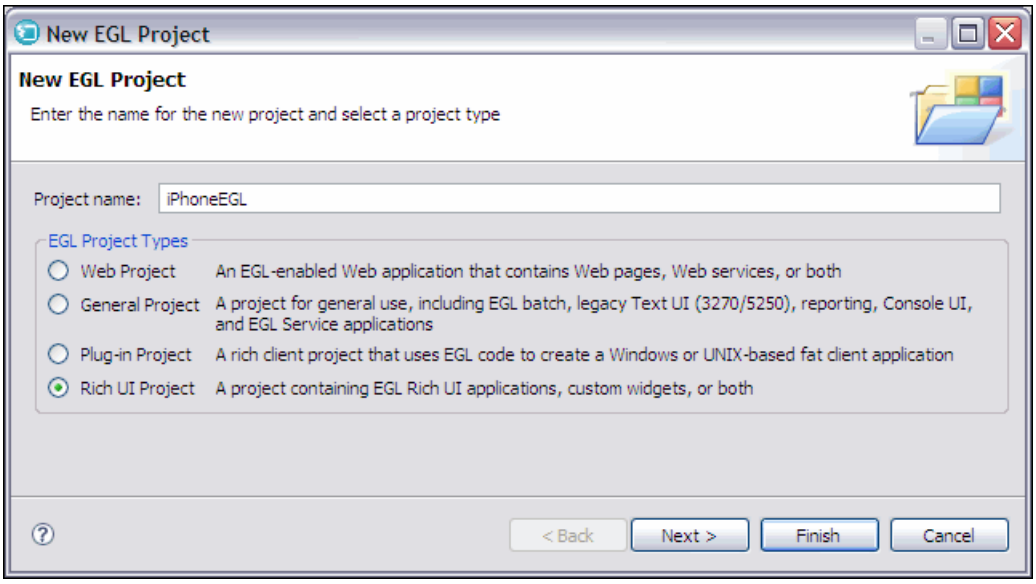


Figure 6-5 New EGL project details

4. Click **Finish.**

   You can explore the complete EGL project structure in the Project Explorer view of the workbench, as shown in Figure 6-6 on page 99.

   The EGL Source folder maintains all source code related to the EGL application. You can create multiple packages to maintain a source code file hierarchy. You will also find one deployment descriptor (egldd) and an EGL build options file (eglbld). These are used to generate java/jee code as well as to specify configuration information.

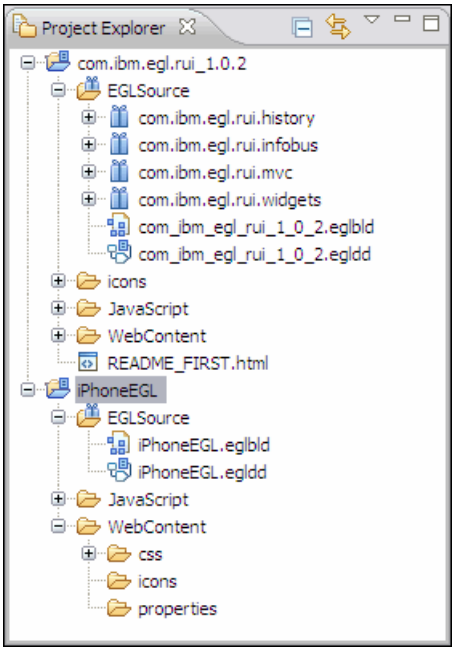


Figure 6-6 EGL Rich UI project structure in Project Explorer view

Your sample project also refers to the `com.ibm.egl.rui` project. This component has the basic widgets, framework, and libraries to be used in the EGL Rich UI project.

5. Now add a Rich UI Handler to the project. In EGL, a handler is a special kind of program with functions that are tied to specific events that occur when someone uses an interface. A Rich UI Handler has these properties:

   – initialUI

     Specifies an array of controls (widgets) that makes up the initial display for the Web page.

   – onConstructionFunction

     Specifies a function to call when the interface is first created.

   – cssFile

     Specifies a cascading style sheet (CSS) to assign to the file. By default, EGL creates a CSS with the same name as the Rich UI Handler, and defines a few basic styles. To customize the appearance of the page, you can modify this file or assign a different one to the cssFile property.

6. Each EGL Rich UI program has one or more handlers. To add a handler, expand the **iPhoneEGL** project in the Project Explorer and right-click the **EGL Source** folder. Select **New** → **Rich UI Handler** from the pop-up context menu, as shown in Figure 6-7 on page 100.

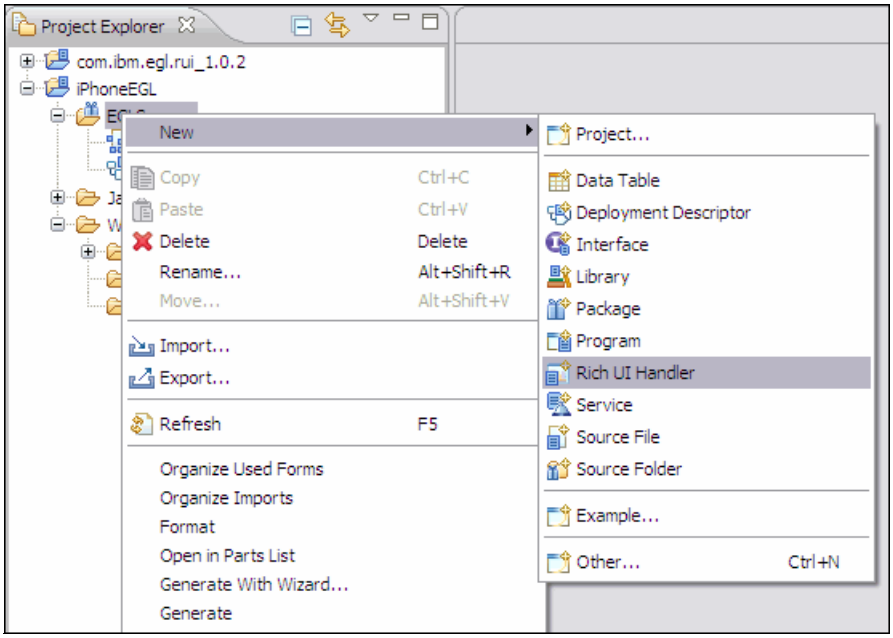


Figure 6-7 Adding a new Rich UI Handler to an EGL project

7. Now update the Package and EGL source file name details in the New Rich UI Handler part pop-up window (Figure 6-8), as follows:

Source folder	Let it default
Package	itso.redbooks
EGL Source file name	CustomerAddHandler

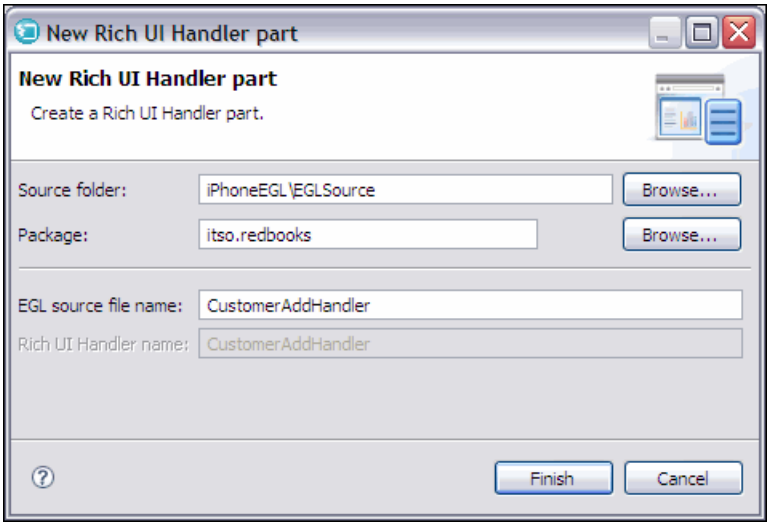


Figure 6-8 Details for new EGL Rich UI Handler

You will see a new `CustomerAddHandler.egl` file added to the `itso.redbooks` package in the EGLSource folder, as shown in Figure 6-9 on page 101.

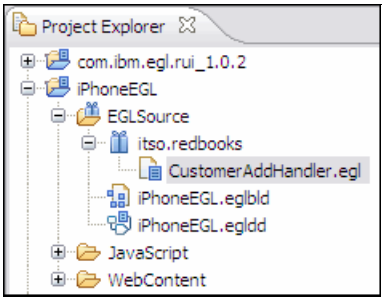


Figure 6-9 CustomerAddHandler added to the EGL project structure

8. Double-click **CustomerAddHandler.egl** to open the EGL editor for the handler code and switch to the **Source** view to look at the source code of handler, as shown in Figure 6-10.

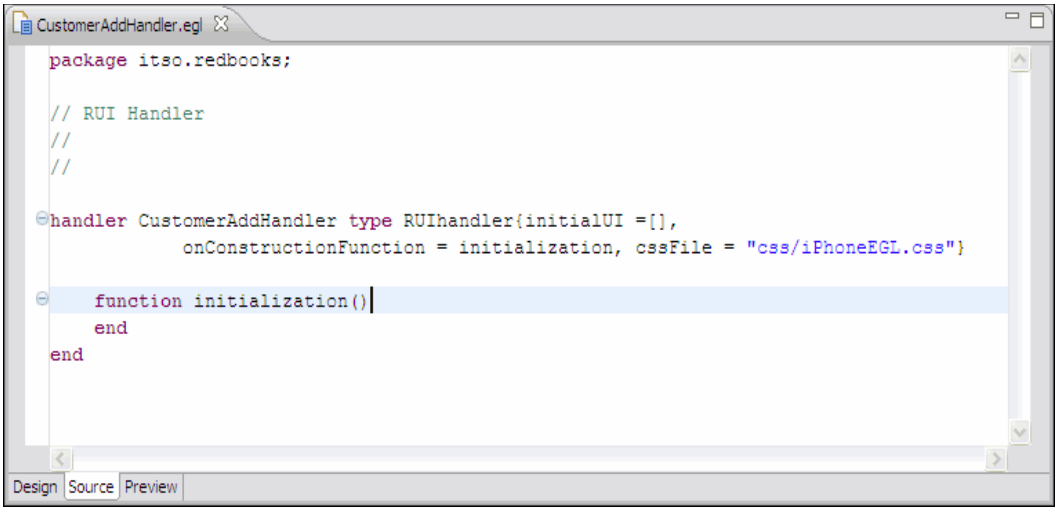


Figure 6-10 Source code view of the CustomerAddHandler.egl source file

9. Change the css file name to iPhoneEGL.css, as shown in Figure 6-10.
10. Double-click **iPhoneEGL.css** available in **iPhoneEGL** → **WebContent** → **css.** The CSS file will now open in the CSS Designer, as shown in Figure 6-11.

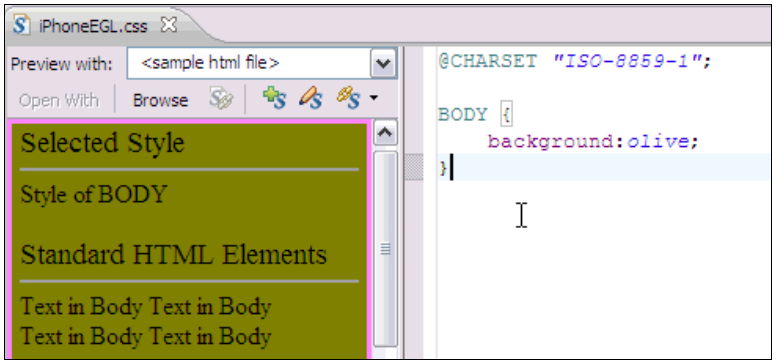


Figure 6-11 CSS file

11. You can update the content of the CSS file to change the background color, as follows:

```
BODY {
   background:olive;
```

```
}
```

As an effect, notice that the background color of `CustomerAddHandler` changes to Olive as specified in the CSS file.

12. You can easily create Rich UI components using the EGL Rich UI perspective and the drag-and-drop visual editor. For this example, try to create:
    - An entry field for customer number
    - Six non-modifiable fields where you will load the data returned from CICS
    - A button that will invoke the CICS Web service

    In the EGL Rich UI:
    - The entry field is a widget called `com.ibm.egl.rui.widgets.TextField`
    - The non-modified field is a widget called `com.ibm.egl.rui.widgets.TextLabel`
    - The button a widget called `com.ibm.egl.rui.widgets.Button`

    You can simultaneously edit the same EGL code using the Visual Rich UI Editor (Design View) and the standard EGL Rich UI Editor (Source View). Using the workbench capabilities, you can rearrange your panel as shown in Figure 6-12.

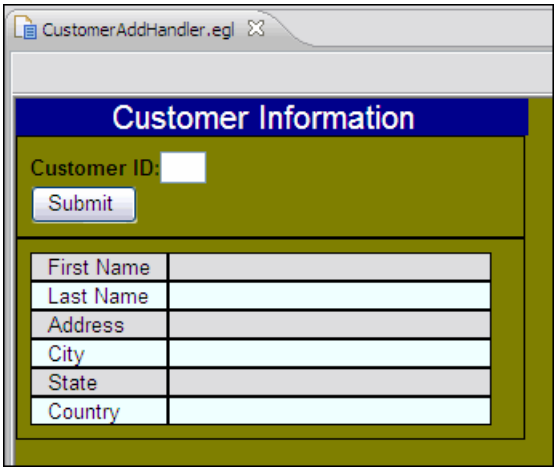


Figure 6-12 Preview of CustomerAddHandler graphical layout

You can also check the Outline view (Figure 6-13 on page 103), It shows a complete structure of the Handler's graphical layout. So, for this example, you can see a Page Title (HTML), first box with Customer ID Label, Customer ID Field, and Submit button, and finally a second box with 12 different labels to show results.

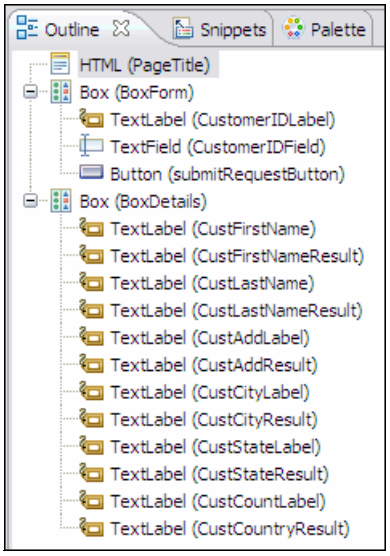


Figure 6-13 Outline of Handler graphical layout

You can use the Properties view for setting up different settings of visual components such as HTML (Title), labels, text fields, or buttons, as shown in Figure 6-14.

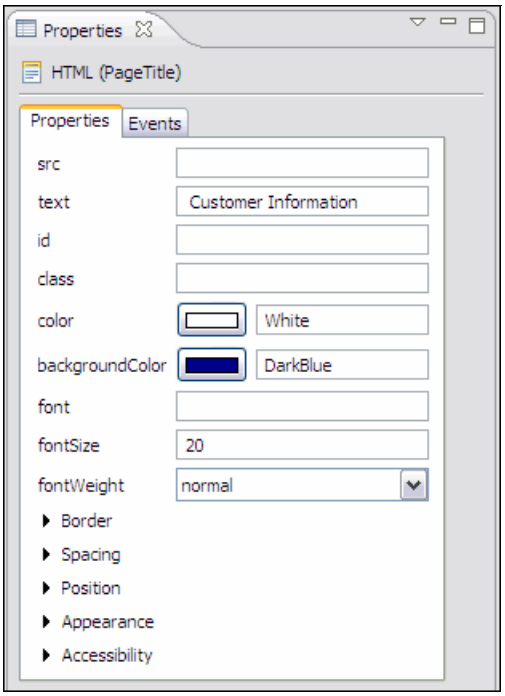


Figure 6-14 Properties view for the HTML component on CustomerAddHandler

13. All changes made in the Properties view automatically reflect in the EGL code. Example 6-1 on page 104 shows the sample code used in our example for the Handler declaration, HTML (title), first box for customer ID form, and widgets declared inside that box.

Example 6-1 Handler declaration and sample widgets code

```
handler CustomerAddHandler type RUIhandler{initialUI =[PageTitle, BoxForm,
BoxDetails ], onConstructionFunction = initialization, cssFile =
"css/iPhoneEGL.css"}
    PageTitle HTML{text = " Customer Information", backgroundColor =
"DarkBlue", fontWeight = "normal", color = "White", fontSize = "20", width =
"260", paddingLeft = 60};

    BoxForm Box{padding = 8, columns = 2, children =[CustomerIDLabel,
CustomerIDField, submitRequestButton ], width = "300", borderColor = "Black",
borderWidth = 1, borderStyle = "double"};

    CustomerIDLabel TextLabel{text = "Customer ID:", fontWeight = "bold",
backgroundColor = "Olive", paddingTop = 3};
    CustomerIDField TextField{width = 25};
    submitRequestButton Button{text = "Submit "};
```

14. Similarly, add another box and 12 labels inside that box to display the results returned from calling the Web service on CICS. Example 6-2 shows the sample code inserted.

Example 6-2 Source code for second box to display results returned from CICS Web service

```
BoxDetails Box{padding = 8, children =[CustFirstName, CustFirstNameResult,
CustLastName, CustLastNameResult, CustAddLabel, CustAddResult, CustCityLabel,
CustCityResult, CustStateLabel, CustStateResult, CustCountLabel,
CustCountryResult ], columns = 2, backgroundColor = "Olive", borderWidth = 1,
borderColor = "Black", borderStyle = "double", width = "300", visibility =
"hidden"};

CustFirstName TextLabel{text = "First Name", backgroundColor = "Gainsboro",
borderWidth = 1, borderColor = "Black", borderStyle = "double", paddingLeft =
10, width = "80"};

CustFirstNameResult TextLabel{width = "200", borderColor = "Black",
backgroundColor = "Gainsboro", borderWidth = 1, borderStyle = "double",
paddingLeft = 5};

CustLastName TextLabel{text = "Last Name", backgroundColor = "Azure",
borderWidth = 1, borderColor = "Black", borderStyle = "double", paddingLeft =
10, width = "80"};

CustLastNameResult TextLabel{backgroundColor = "Azure", borderColor = "Black",
borderWidth = 1, borderStyle = "double", width = "200", paddingLeft = 5};

CustAddLabel TextLabel{text = "Address",...};
CustAddResult TextLabel{backgroundColor = "Gainsboro",...};

CustCityLabel TextLabel{text = "City",...};
CustCityResult TextLabel{backgroundColor = "Azure", ...};

CustStateLabel TextLabel{text = "State",...};
CustStateResult TextLabel{backgroundColor = "Gainsboro",...};

CustCountLabel TextLabel{text = "Country", ...};
CustCountryResult TextLabel{backgroundColor = "Azure",...};
```

15. You can right-click the editor (Source view) and select **Format** to format the source code.

Importing the WSDL file of the CICS Web service

The interface to the CICS Web service is described in a WSDL file. You need to import the WSDL file of your CICS Web service into the EGL project to generate the skeleton client interface and then configure the EGL client to talk to the WSDL client interface.

1. To import the WSDL file in your project, right-click the **iPhoneEGL** project in Project Explorer and select **Import** from the pop-up context menu.
2. Expand **General** → **File System** in the pop-up window and click **Next >**.
3. Browse to the folder containing the WSDL file in the file system, select the check box for the WSDL file to use and then click **Finish**. In our example we are using a WSDL file called `CUSTINQ.wsdl`, representing a CICS Web service for the CUSTINQ program.

   You will see the WSDL file imported in the iPhoneEGL project structure, as shown in Figure 6-15.

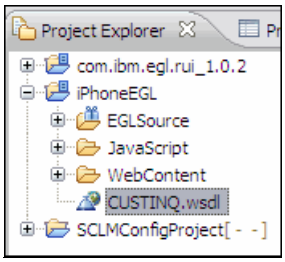


Figure 6-15 Imported WSDL file for CICS Web service in the project structure

Service invocation in a Rich UI is always asynchronous, which means that the requester (the Rich UI client) continues running without waiting for a response from the service. The user can still interact with the user interface while the Rich UI Handler waits for the service to respond. After the invocation, the service does some task and, in most cases, responds to the EGL runtime, which in turn invokes a Rich UI function that you code, called a *callback* function.

To invoke a service from a Rich UI, you need to create an EGL interface part that includes properties that indicate how the service is accessed at run time. You then create a variable based on the interface part and use the variable in a call statement. The call statement includes the details necessary for the EGL run time to issue a callback.

We now show how to generate the client interface for the WSDL so that the EGL client application can call the Web service by using that interface.

4. To create the client interface, right-click the WSDL file and select **EGL Services** → **Create EGL Client Interface...** from the pop-up context menu, as shown in Figure 6-16 on page 106.

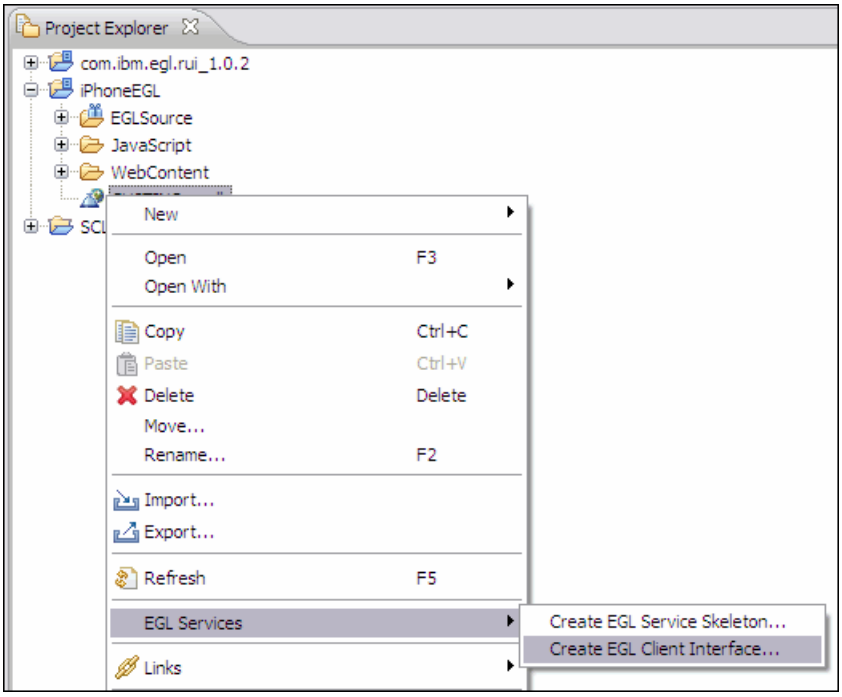


Figure 6-16 Creating an EGL Client interface from a WSDL file

A pop-up window appears with confirmation of the new EGL interface. Select the check box for **CUSTINQPortType** and also select the check box for **Create Web services Client Bindings**, as shown in Figure 6-17.

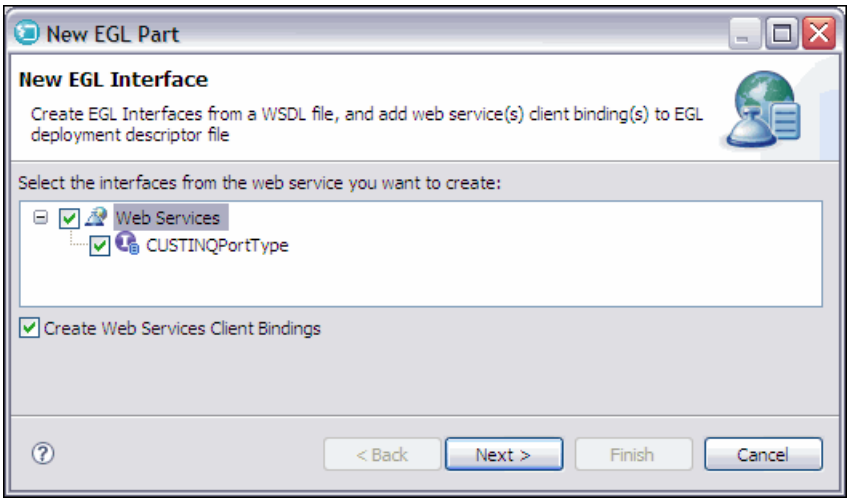


Figure 6-17 EGL Client interface type generation with Web Services client bindings

5. Click **Next >.**

   A new EGL interface is created in a new EGL source file. So here you can update the name of the package and source file, if required. The interface name is CUSTINQPortType and the operation name to call the Web service is CUSTINQOperation. Here the package name is `files.target` and the EGL source file name is CUSTINQ. See Figure 6-18 on page 107.

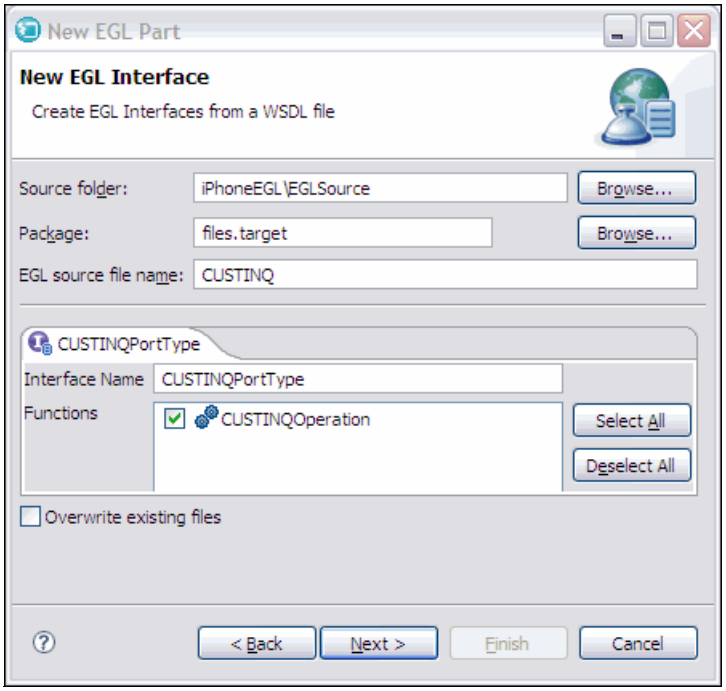


Figure 6-18 New EGL interface generation - source file name and package inputs

6. Click **Next >.**

   The following step, shown in Figure 6-19, is required to map the WSDL client bindings to the EGL deployment descriptor. The deployment descriptor in our case is iPhoneEGL. It maintains the information needed by the service client to call the Web service.

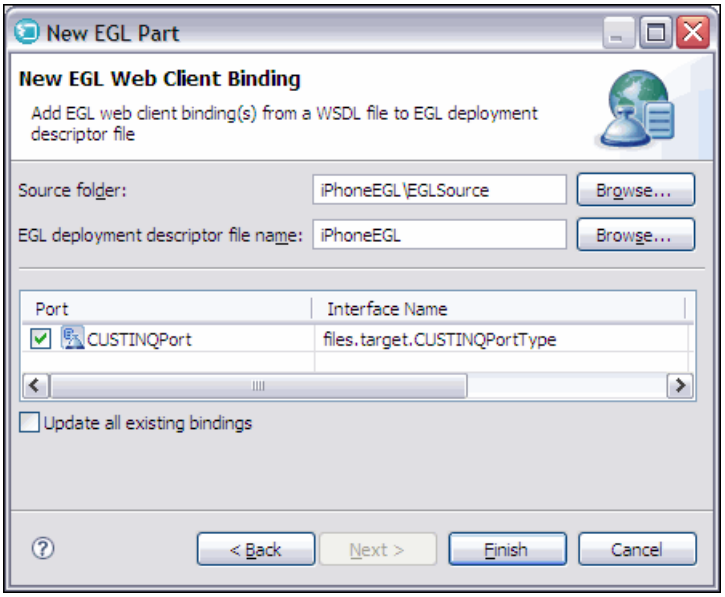


Figure 6-19 EGL Web client binding - from WSDL to deployment descriptor

7. Click **Finish.**
8. You can now double-click **iPhoneEGL.egldd** to open the deployment descriptor, The information with respect to the WSDL file is configured in the Service Client Bindings section, as shown in Figure 6-20.

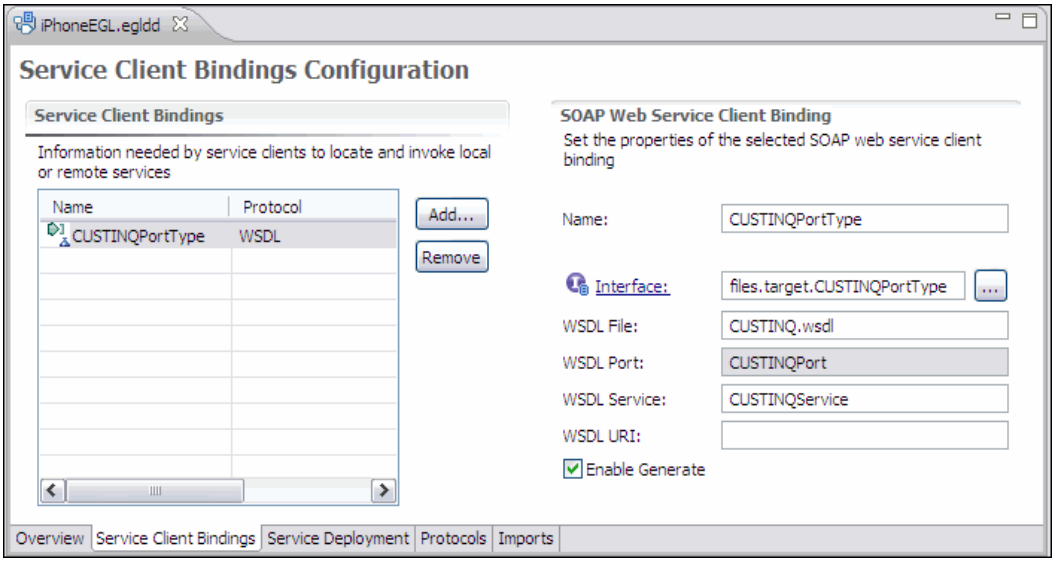


Figure 6-20 WSDL binding for EGL deployment descriptor

9. You can now also explore the different EGL source folders added for the WSDL interface definitions in the Project Explorer. It includes input parameter definition (request), output parameter definition (response) and interface definition to be used by the EGL Client. See Figure 6-21.

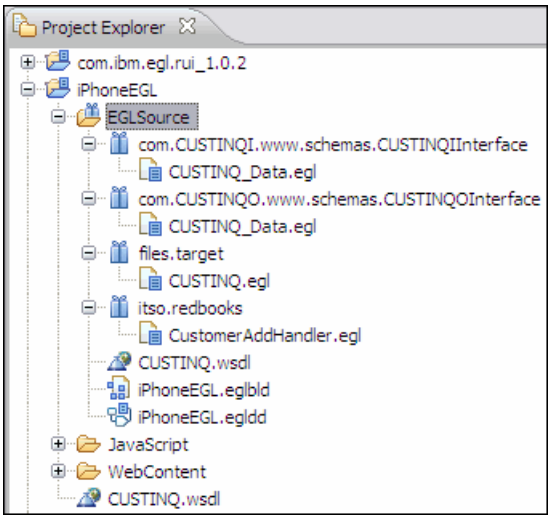


Figure 6-21 EGL source files generated based on the WSDL file

10. Double-click the **CUSRINQ_Data.egl** source file in the `com.CUSTINQI.www.schemas.CUSTINQIInterface` package. It defines the parameters to hold the input to make Web service requests (see Example 6-3 on page 109).

Example 6-3 Web service client interface input parameter (request)

```
package com.CUSTINQI.www.schemas.CUSTINQIInterface;

dataItem CustNo int{}
end

record DFHCOMMAREA{}
  CustNo CustNo;
end
```

Similarly, CUSTINQO.www.schemas.CUSTINQOInterface defines the resulting response from the Web service, as shown in Example 6-4.

Example 6-4 Response data definition for the Web service

```
package com.CUSTINQO.www.schemas.CUSTINQOInterface;

dataItem CustNo int{}
end

dataItem LastName string{}
end

dataItem City string{}
end

dataItem FirstName string{}
end

dataItem Address1 string{}
end

dataItem State string{}
end

dataItem Country string{}
end

dataItem RetCode smallInt{}
end

record DFHCOMMAREA{}
    CustNo CustNo;
    LastName LastName;
    FirstName FirstName;
    Address1 Address1;
    City City;
    State State;
    Country Country;
    RetCode RetCode;
end
```

The interface code is generated in the files.target package and the function to be called from the EGL Web service client application would be CUSTINQOperation with the appropriate input parameters. The function call will then return the output data structure as discussed above. The source code of our example is shown in Example 6-5 on page 110.

Example 6-5 EGL Client interface definition

```
package files.target;

//@webBinding{wsdlLocation="CUSTINQ.wsdl", wsdlPort = "CUSTINQPort",
wsdlService = "CUSTINQService"}

interface CUSTINQPortType{@xml{name = "CUSTINQPortType", namespace =
"file://target.files"}}

function CUSTINQOperation(DFHCOMMAREA
com.CUSTINQI.www.schemas.CUSTINQIInterface.DFHCOMMAREA in)
returns(com.CUSTINQO.www.schemas.CUSTINQOInterface.DFHCOMMAREA){@xml{name =
"CUSTINQOperation"}};
end
```

Making the client application talk to the EGL client interface

We now add three methods to make the EGL application client talk to the CICS Web service or EGL skeleton client generated, based on the WSDL file.

The first method (`getCustomerDetails`) is to call the WSDL client interface from `CustomerAddHandler`; we perform the following in this method:

- Pick up the value of the customer ID from the text field.
- Create a connection to the Web service through client binding.
- Call the service and get the response.
- Pass the response to another method for displaying the information in a page.
- In case of some error or problem, call an exception handler method.

This method is shown Example 6-6.

Example 6-6 Method to call a CICS Web service through the WSDL client binding

```
function getCustomerDetails(e event in)
    customerid string = CustomerIDField.text; // get input type
      serv CUSTINQPortType{@BindService};
   inputMsg com.CUSTINQI.www.schemas.CUSTINQIInterface.DFHCOMMAREA;
   inputMsg.CustNo = customerid;
   outputMsg com.CUSTINQO.www.schemas.CUSTINQOInterface.DFHCOMMAREA;
   call serv.CUSTINQOperation(inputMsg) returning to displayInformation
onException handleGetCustomersError;

   end
```

Once we have a response from the Web service, we pass the output result to another method called `displayInformation` for displaying information on the `CustomerAddHandler` page. We bind each result value returned to an associated label field. See Example 6-7 on page 111.

Example 6-7 Displaying information or load data in text labels to display

```
function displayInformation(outputMsg
com.CUSTINQO.www.schemas.CUSTINQOInterface.DFHCOMMAREA in)
        CustFirstNameResult.text = outputMsg.FirstName;
        CustLastNameResult.text = outputMsg.LastName;
        CustAddResult.text = outputMsg.Address1;
        CustCityResult.text = outputMsg.City;
        CustStateResult.text = outputMsg.State;
        CustCountryResult.text = outputMsg.Country;
    end
```

In case of an error or exception (for example, a connection problem) the `handleGetCustomersError()` method is called and an error message is displayed to the user. See Example 6-8.

Example 6-8 Exception handling in case of some error

```
private function handleGetCustomersError(exp AnyException in)
        e ServiceInvocationException = exp;
        writeStdout("An error occurred. " + e.message + " " + e.detail3);
        // Display error message
        CustFirstNameResult.color = "red";
        CustFirstNameResult.text = "Error while invoking the service. The server
may be down or not responding.";
    end
```

We now associate the Submit button widget on `CustomerAddHandler` to call the `getCustomerDetails` method. And we can do so by selecting the button and setting the value of the onClick event in the events section of the Properties view. We set it to **getCustomerDetails** from the drop-down list, as shown in Figure 6-22.

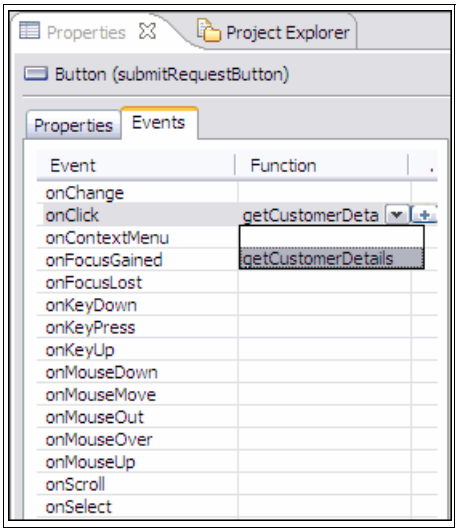


Figure 6-22 Setting a button to call a method on CustomerAddHandler

We now save the CustomerAddHandler.egl and switch to preview mode. The Web service can now be tested by entering a customer ID in the text field and pressing **Submit.** The EGL `CustomerAddHandler` will make a connection to the Web service, pull the information from CICS, and display the results shown in Figure 6-23 on page 112.

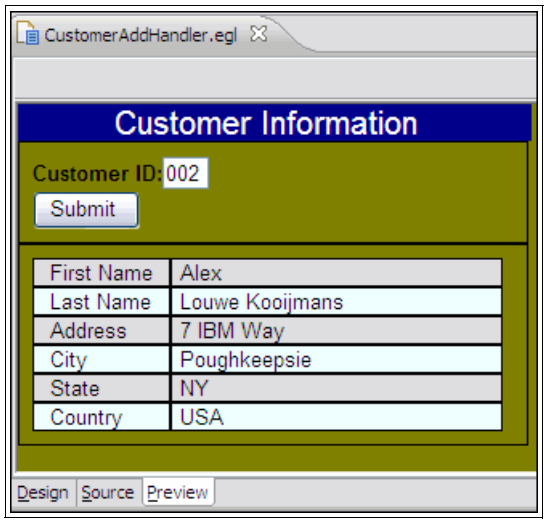


Figure 6-23 Preview of EGL Client application - results of a Web service call

Generating the application for deployment

Now that the EGL Client application calling a CICS Web service has been tested successfully, we can focus on generating a JEE application to host it on WebSphere Application Server. Follow these steps:

1. Right-click the **iPhoneEGL** project in Project Explorer and select **Deploy Rich UI Application** from the context menu. It will ask you to change the Rich UI generation mode to Deployment (Figure 6-24).

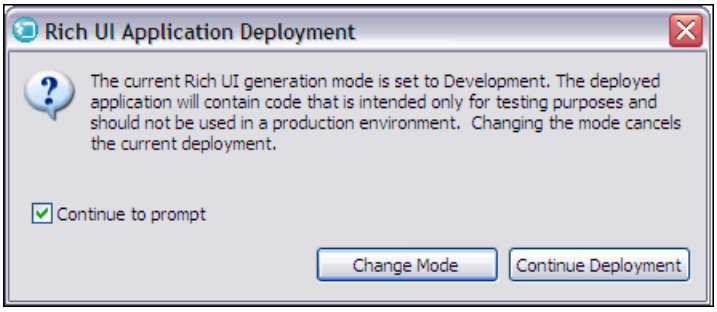


Figure 6-24 Rich UI Application Deployment mode

2. Click **Change Mode,** change the mode to **Deployment** in the pop-up preferences window as shown in Figure 6-25 on page 113, and click **OK.**

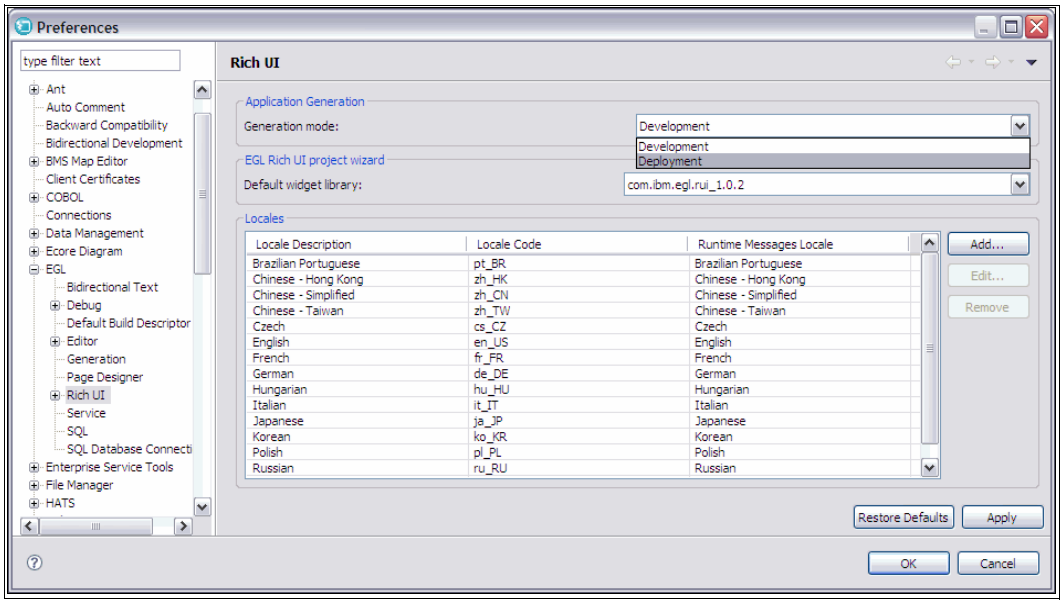


Figure 6-25 Changing Rich UI application generation mode to Deployment

3. A pop-up window will then ask for the Source Project and Rich UI Handler you want to generate source for and the target application server. In our case, we used the following values, as also shown in Figure 6-26 on page 114:

   Source Project iPhoneEGL

   Rich UI Handler CustomerAddHandler.egl

   Deployment Solution WebSphere Application Server Deployment

   Also, select the check box **Save deployment configuration.**

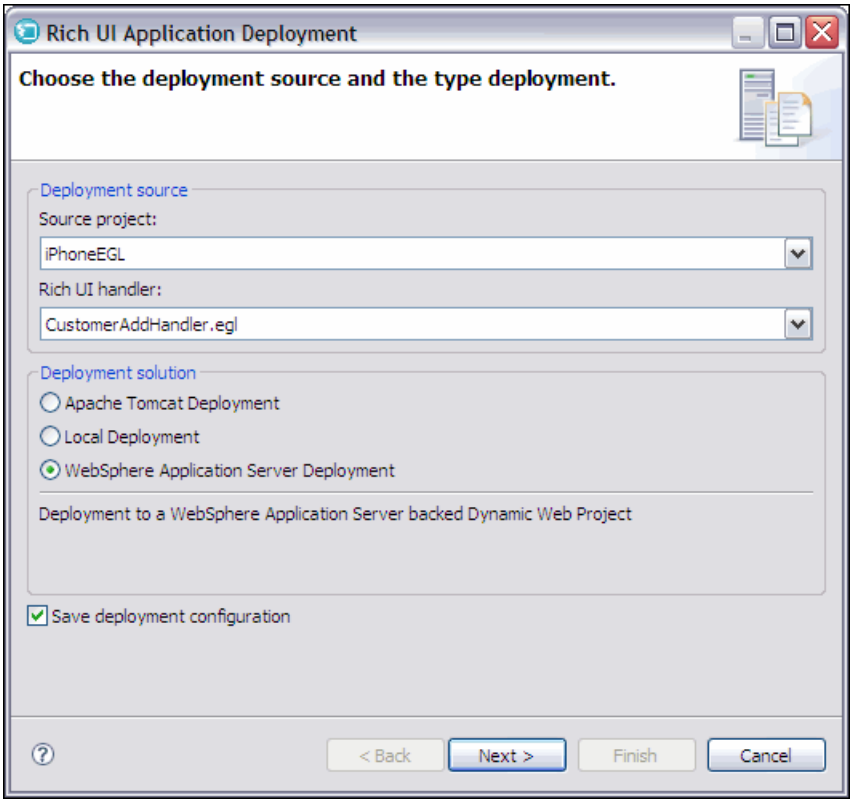


Figure 6-26 Rich UI Application Deployment settings

4. Click **Next >.**
5. Specify a new Web project with these settings shown in Figure 6-27 on page 115:

Project Name	iPhoneJEE
Runtime Server	WebSphere Application Server 7.0
Context Root	iPhoneJEE
Language options	English

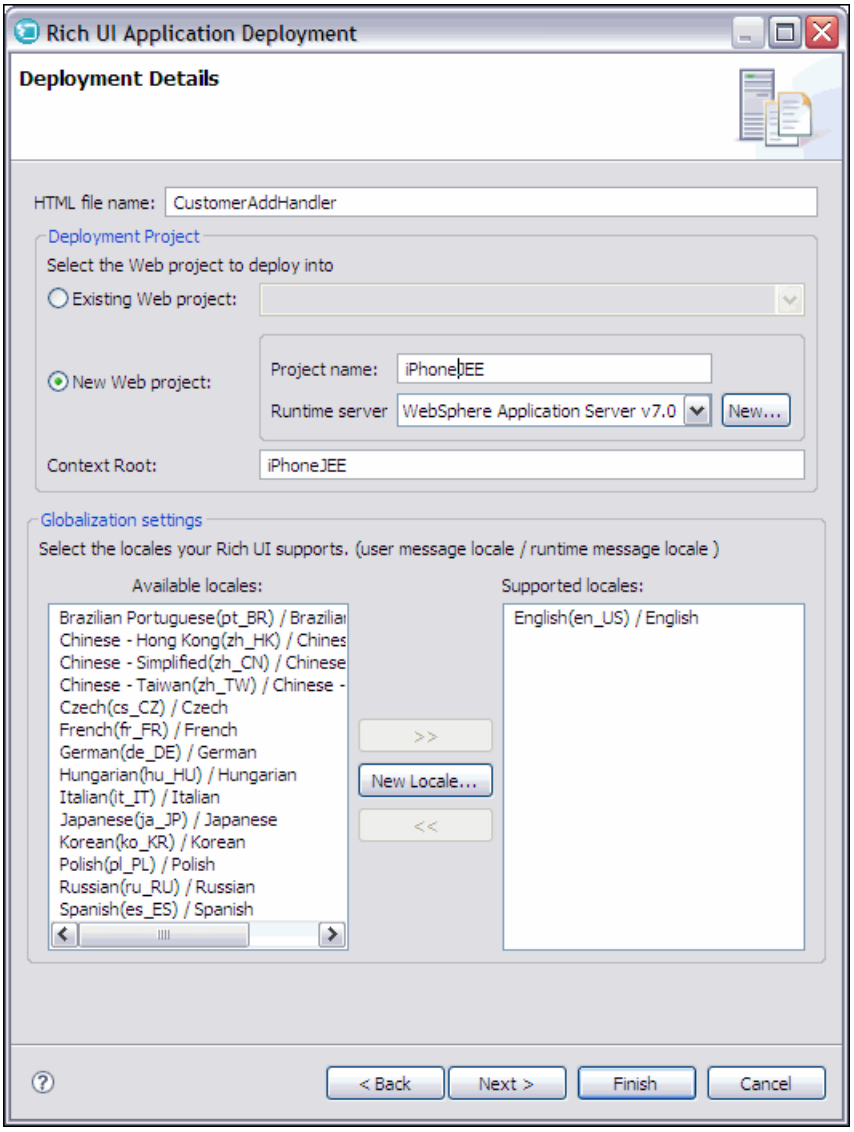


Figure 6-27 Configuring options to generate a JEE Web application

6. Click **Next >.**
7. Select all the artifacts to be generated, including stylesheets, images, icons, as well as properties files. Click **Finish**. See Figure 6-28 on page 116**.**

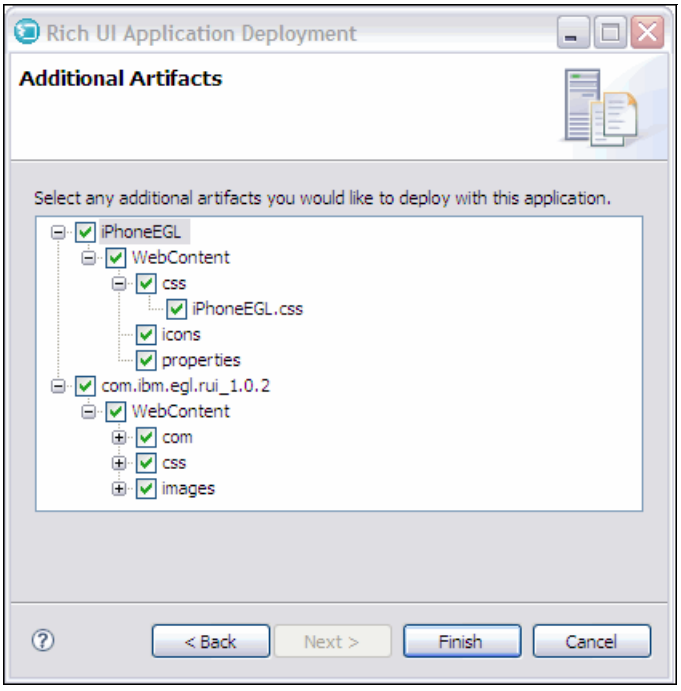


Figure 6-28 Artifacts to be generated because of RichUI Application deployment

8. Once all the artifacts related to the JEE project are generated, you will see a confirmation message. Click **OK**, as shown in Figure 6-29**.**

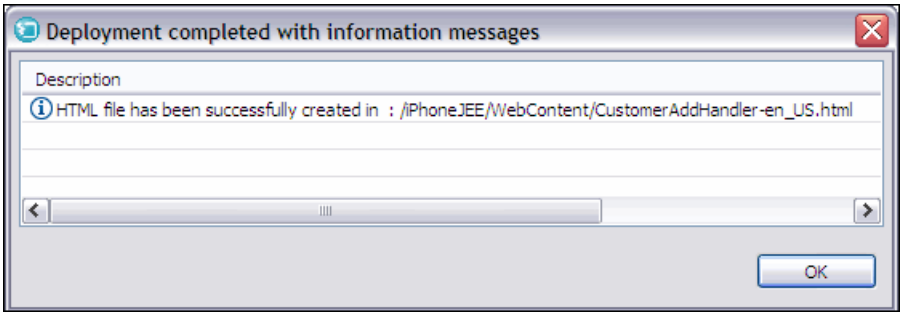


Figure 6-29 Confirmation of successful generation of JEE artifacts from EGL

9. Explore all the artifacts of the iPhoneJEE project in Project Explorer. You will see the CustomerAddHandler-en_US.html page under the WebContent folder, as shown in Figure 6-30 on page 117.
10. Rename CustomerAddHandler-en_US.html to just CustomerAddHandler.html.

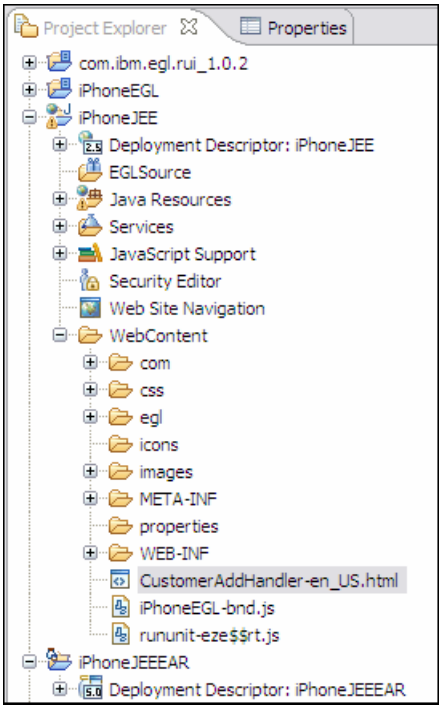


Figure 6-30 CustomerAddHandler-en_US.html file generated in Project Explorer

11. Double-click **CustomerAddHandler.html** to open up the HTML source code in the HTML editor. Add the following html meta tag in the html section of the HTML page, as shown in Example 6-9.

Example 6-9 HTML - Meta tag to be added in the page

```
<meta name="viewport"
content="width=device-width,user-scalable=no">
```

For better display of the HTML page on an Apple iPhone panel, add this line instead; Example 6-10.

Example 6-10 HTML - Meta tag for display in Apple iPhone

```
<meta name="viewport" content="width=320, user-scalable=no">
```

While adding the line in CustomerAddHandler.html you may get a pop-up message as shown in Figure 6-31. Just click **Yes.**

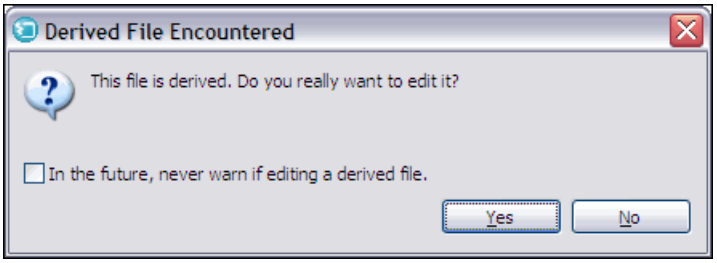


Figure 6-31 Derived file modification confirmation message

The following JavaScript may be needed to display the HTML page as desired.

– In case you want to remove the address bar from the browser and want the application to look like a native Apple iPhone application, you can add the following JavaScript in your HTML page; Example 6-11.

Example 6-11 JavaScript to hide the address bar in the HTML page

```
<script type="application/x-javascript">
    if (navigator.userAgent.indexOf('iPhone') != -1)
    {
        addEventListener("load", function()
        {
        setTimeout(hideURLbar, 0);
        }, false);
    }
    function hideURLbar()
    {
        window.scrollTo(0, 1);
    }
</script>
```

– If you want to add a custom icon for your application when a user bookmarks the application to the home panel, you can do so by adding the following JavaScript in the HTML page; Example 6-12. We have an icon file located in the icons folder with the name Finder.png that will be used as an icon for the application bookmark.

Example 6-12 JavaScript to add a custom icon

```
<link rel="apple-touch-icon" href="icons/Finder.png"/>
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black" />
```

Testing the application in WebSphere Application Server

Now you are set to test your application on WebSphere Application Server.

1. Right-click **CustomerAddHandler.html** and select **Run** → **Run on Server** from the pop-up context menu.
2. Select the server you want to test your application on and click **Next;** Figure 6-32 on page 119.

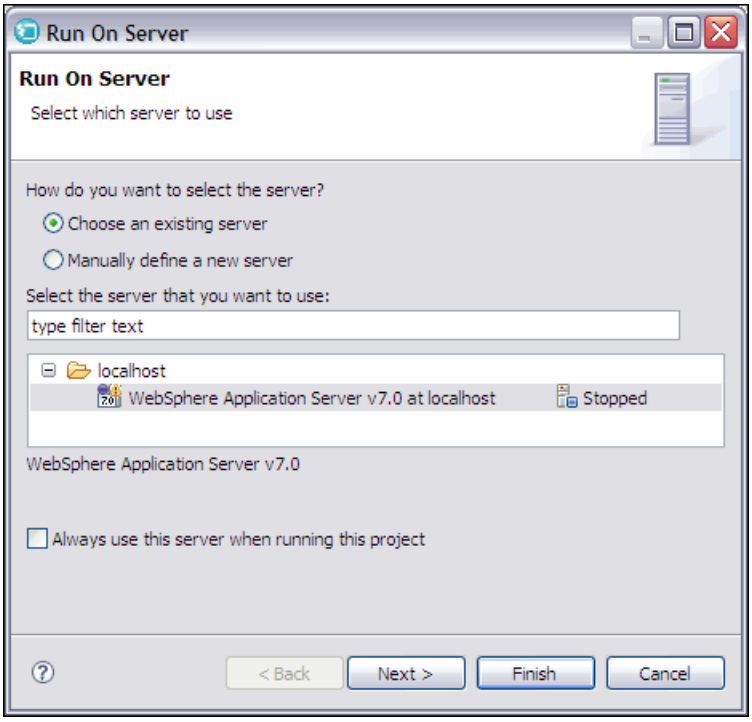


Figure 6-32 Run on server

3. Add your project to Configured projects, as shown in Figure 6-33.

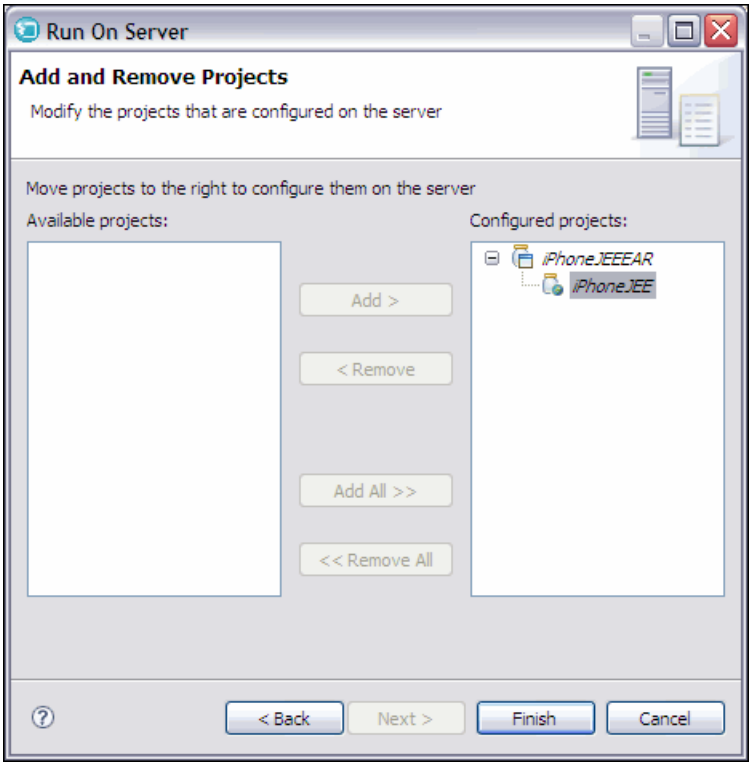


Figure 6-33 Adding your project for publishing it on server

4. Click **Finish**.

   The server will now be started and the application will be published to WebSphere Application Server. See Figure 6-34.

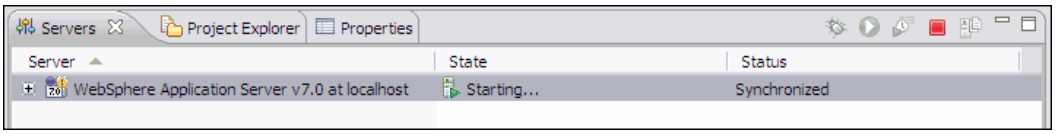


Figure 6-34 Server starting and application publishing

Once the server is up and running you will see CustomerAddHandler.html opening in a browser. You can test the application by inserting a value for customer ID, as shown in Figure 6-35.

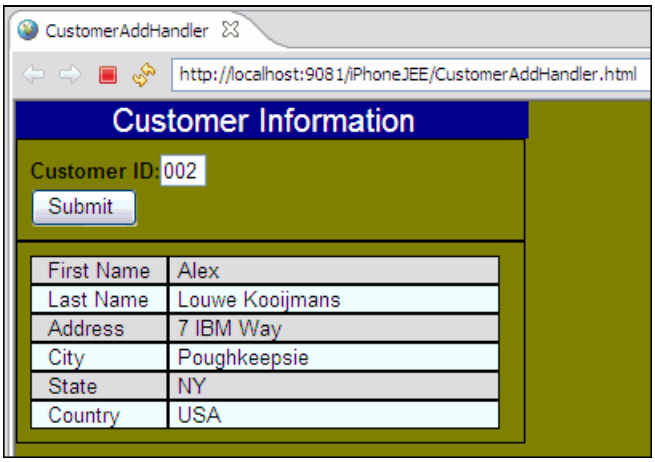


Figure 6-35 Testing of JEE application client on WebSphere Application Server

Exporting the JEE application as EAR file and installing it on a remote server

You can also export the JEE application as an EAR (Enterprise Archive) file and install it on WebSphere Application Server remotely.

1. To export a JEE application, you can right-click the **iPhoneJEEEAR** folder in the Project Explorer and select **export** from the pop-up context menu.
2. Select the export type as EAR and save it to a location on your file system.
3. To install the `iPhoneJEEEAR.ear` on a remote server, you can use the Administrative Console of the remote WebSphere server to install it. Once the application is installed, you can test it on iPhone. A connection to the remote WebSphere Application Server can also be made from RDz directly for JEE application publishing and testing purposes. Refer to Appendix C, “Deployment to WebSphere Application Server on z/OS” on page 197 for details.

Testing on iPhone

Open the browser in iPhone and point it to the address of your application. In our case it is:

```
http://wtscz1.itso.ibm.com:8207/iPhoneJEE/CustomerAddHandler.html
```

You can see the application being loaded in the browser. Figure 6-36 on page 121 shows a panel shot from the iPhone.

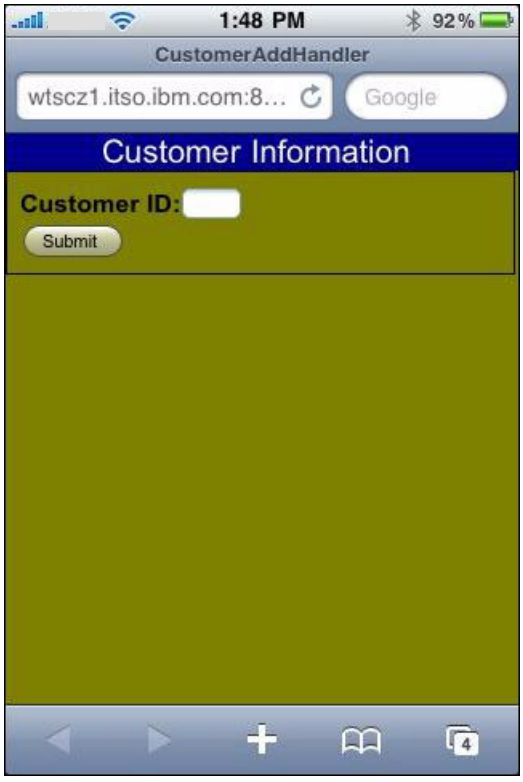


Figure 6-36 Application loaded in the iPhone browser

Insert the customer ID in the text field with the help of the iPhone keypad and touch **Submit**; Figure 6-37 on page 122.

Figure 6-37 Inserting Customer ID in the text field

You can see the result being displayed on the IPhone panel; Figure 6-38.

Figure 6-38 Result of a successful call to CICS Web service

6.5 Key considerations

To develop Web clients for other kinds of smartphones, such as Android and Blackberry, the basic steps will remain the same for EGL, but once JEE code is generated, then platform-specific customization needs to be done to fit the user interface to the specific device.

7

Accessing a CICS Restful Web service from Apple iPhone

This chapter discusses a scenario in which we access a CICS Web service using REST from Apple iPhone in a logical two-tier architecture.

© Copyright IBM Corp. 2010. All rights reserved.

7.1 High-level architecture

The high-level architecture of this scenario is shown in Figure 7-1.

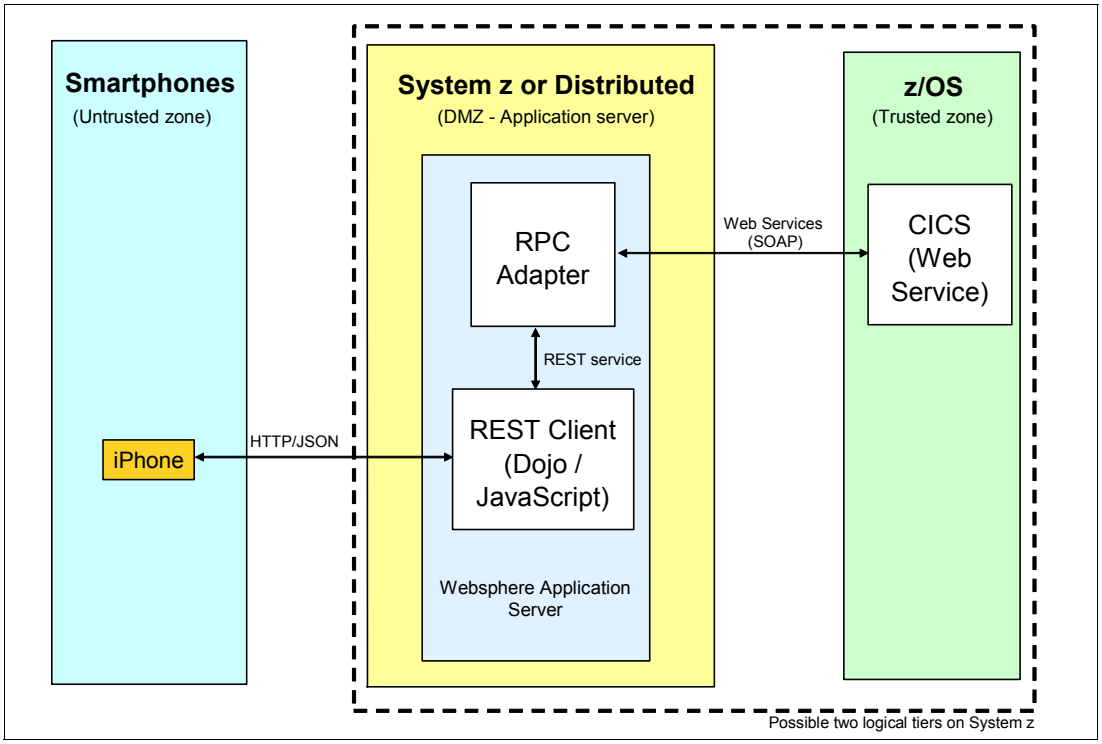


Figure 7-1 Solution architecture - Accessing a CICS RESTful service on z/OS

7.2 Software used

The following software is used to develop, test, and run this scenario:

- Rational Developer for System z (RDz) Version 7.6
- WebSphere Application Server Version 7.0
- Internet Browser: Safari, Internet Explorer, or Firefox

7.3 Back-end requirements

In order to use this scenario with one of your CICS back-end applications you need to have a version of CICS supporting Web Services. Also, you need to have CICS Web Services configured and ready to use. Finally, you need to have the CICS application you wish to use enabled as a Web service. The WSDL file belonging to this CICS Web service is required in the development and testing process of the REST interface.

Creating and deploying a Web service for CICS is not in the scope of this book, but has been documented already in numerous other publications. A good starting point is:

http://www.redbooks.ibm.com/abstracts/sg247126.html?Open

7.4 Step-by-step instructions

The code developed for this scenario is available for download as an EAR file as well as a Project Interchange File. Refer to Appendix D, “Additional material” on page 203 for instructions.

7.4.1 Getting started

To get started, follow these steps:

1. Open Rational Developer for system z (RDz). While opening RDz it will prompt you for a workspace name. You can simply choose any folder on the local workstation for this. *Workspace* is a place where all artifacts related to your project will be stored.
2. Switch to the Web perspective by selecting **Window** → **Open Perspective** → **Other** and choose **Web** from the pop-up window.

   In this scenario we create a new Dynamic Web Project to extend a CICS Web service as a REST service.
3. To create a new Dynamic Web Project, select **File** → **New** → **Dynamic Web Project** from the RDz workbench toolbar.
4. Update the following in the pop-up dialog, as also shown in Figure 7-2 on page 128:

   Project Name CicsRESTI

   Target Runtime Websphere Application Server 7.0

   Dynamic Web Module Version 2.5

   Also, tick the check box **Add project to an EAR** and specify `CicsRESTIEAR` as EAR file name.

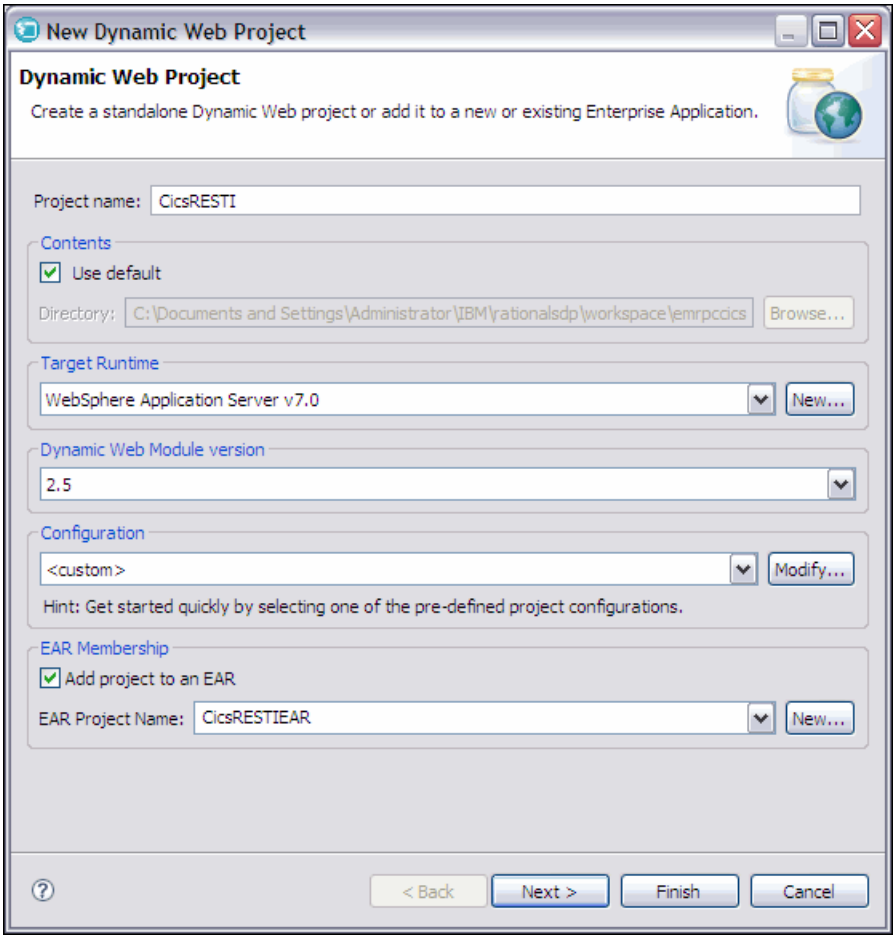


Figure 7-2 Creating a Dynamic Web Project

5. Next, you need to change the default configuration for WebSphere Application Server. Click **Modify...** adjacent to the Configuration text field.
6. This will pop up the Project Facets dialog in which you need to select **Web 2.0** from the list. Click **OK.**
7. You will return back to the New Dynamic Web Project dialog and the configuration text box will be updated with **<custom>.**

   By selecting Web 2.0 from the list you are adding Web 2.0-specific libraries to the project that include an RPC Adapter, which is also known as *Web remoting*.

Note: *Web remoting* is a pattern that provides support for JavaScript or client code to directly invoke server logic. This pattern provides the ability to invoke Java methods from JavaScript.

The IBM implementation for Web remoting is referred to as the *IBM RPC Adapter*. The RPC Adapter is designed to help developers create command-based services quickly and easily in a manner that complements the programming styles applicable to AJAX applications and other lightweight clients. Implemented as a generic servlet, the RPC Adapter provides an HTTP interface to registered Java Beans.

Web remoting provides a lightweight Web endpoint that can expose methods of Java EE assets (EJB, POJO, and Web service proxies). Web remoting is easily invoked from AJAX applications using JSON or XML formats, and supports HTTP GET/POST mapping for methods. It is enabled through simple configuration options without requiring modifications to the original Java objects, EJB, or Web Services.

8. Click **Next** three times and you will see a window with Web 2.0 Server-Side Technologies - RPC Adapter configuration options. Select the check box **Add Servlet to Deployment Descriptor** and also keep the **Servlet mapping** location as default to **/RPCAdapter/*** (Figure 7-3).

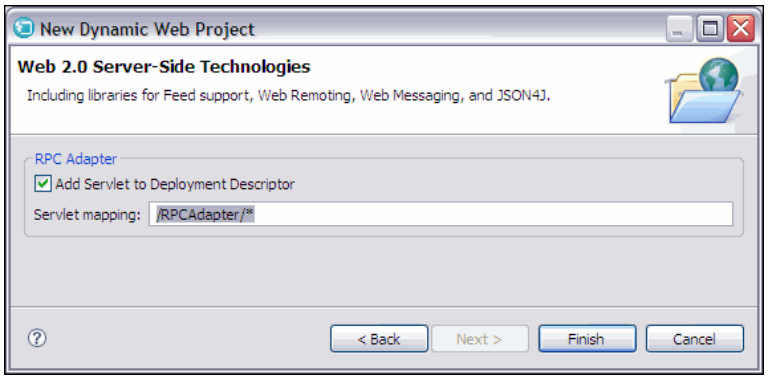


Figure 7-3 RPC Adapter configuration

9. Click **Finish.**

A Dynamic Web Project will be created in the workspace and you can browse the contents in the Project Explorer view, as shown in Figure 7-4.

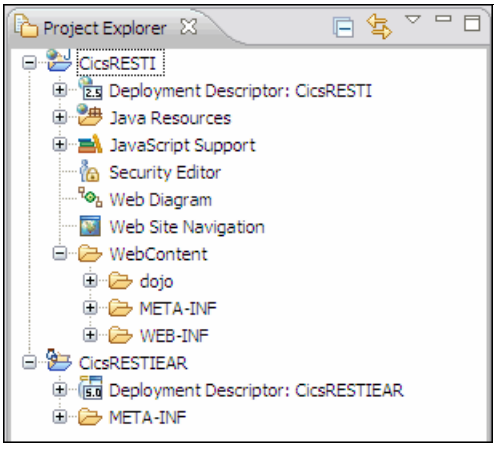


Figure 7-4 Created Dynamic Web Project in the Project Explorer

7.4.2 Importing the WSDL file

To be able to build the client, you need a WSDL file describing the service interface. This WSDL file needs to be added to the project.

1. To import a WSDL file in the Web project recently created, right-click the **CicsRESTI** project folder in the Project Explorer and select **Import** → **Import ...** from the pop-up context menu.
2. Expand **General** → **File System** from the pop-up Import window and click **Next >**. Browse for the WSDL file in the file system (CUSTINQ.wsdl in our case) and click **Finish.** You will see the CUSTINQ.wsdl file added to the project, as shown in Figure 7-5.

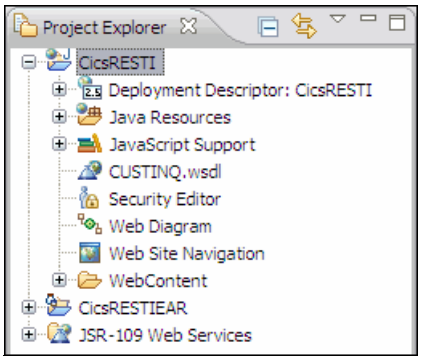


Figure 7-5 WSDL file added to the project

7.4.3 Creating a Web service client to be exposed as REST service

In the following steps we show how you can quickly create a REST service based on the WSDL just imported.

1. Right-click the **CUSTINQ.wsdl** file in the Project Explorer and select **Web Services** → **Generate Client** from the context menu. This will result in the Web Service Client dialog, shown in Figure 7-6 on page 131.

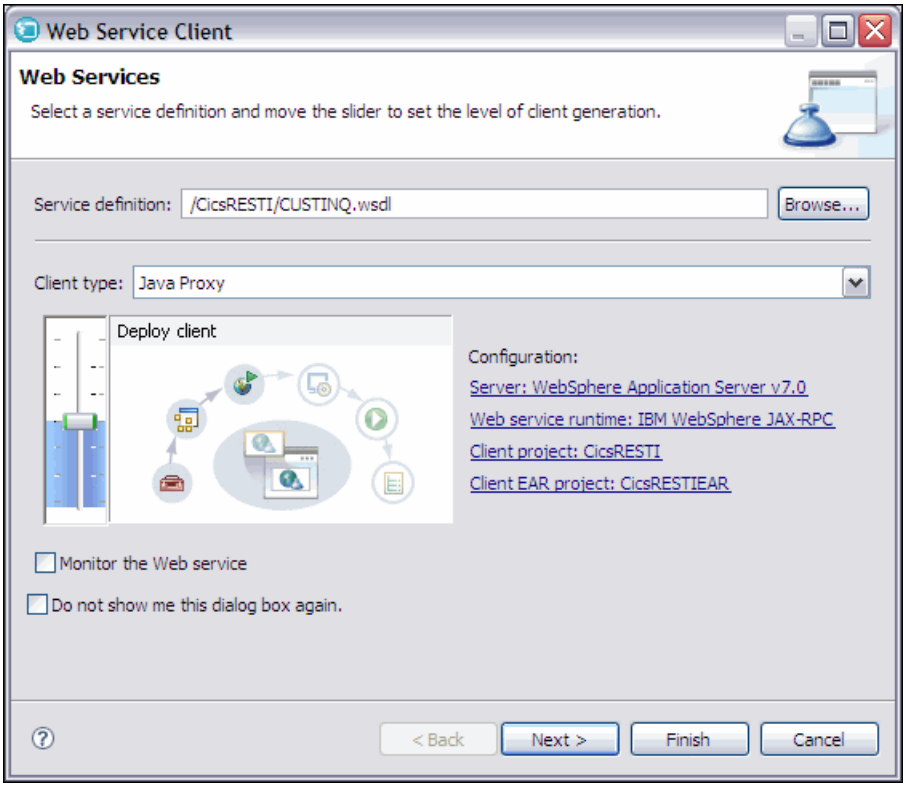


Figure 7-6 Generating a Web Services client

2. Click **Finish.**

   You may be prompted with a warning message, as shown in Figure 7-7.

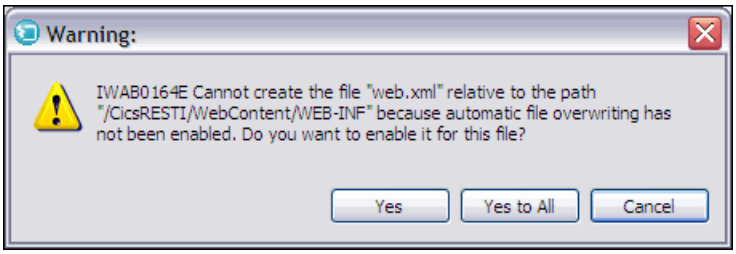


Figure 7-7 Warning message while generating a Web Service client

3. Just click **Yes to All.**

   Now Java skeleton code is generated to call a Web service from the Dynamic Web Project. It basically adds a few packages and Java source files to the project structure. You can browse for packages and source files in the Project Explorer view.

   You can notice that packages with names such as `com.CUSTINQI.www`, `com.CUSTINQO.www` and `files.target` have been added. Each package has some Java source files. Package `com.CUSTINQI.www` contains input parameters, its serializer, deserializer, and helper classes. Package `com.CUSTINQO.www` contains output parameters, its serializer, deserializer, and helper classes. And package `files.target` contains classes for the Web service proxy and stub.

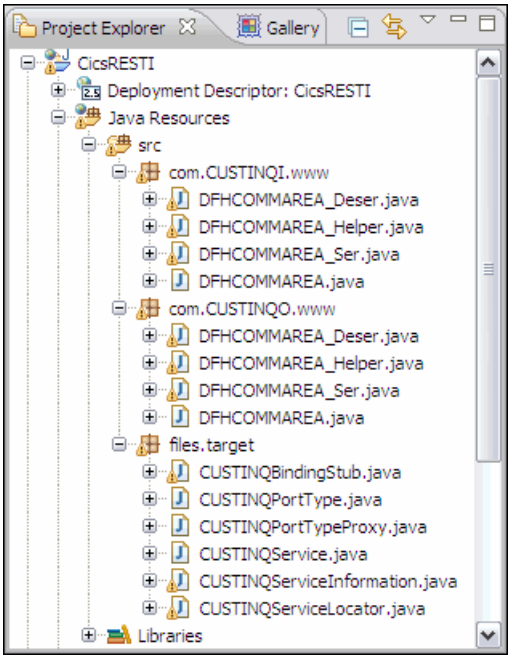


Figure 7-8 Project Explorer with generated Web Services client packages

We now show how to add a new class as a Web service client and expose that class for REST interaction.

4. To add a new class, right-click the **CicsRestI project** in Project Explorer and select **New** → **Class** from the pop-up menu.
5. Update the Package name to `files.target` and the class name to `WsRestI`, as shown **in** Figure 7-9 on page 133.

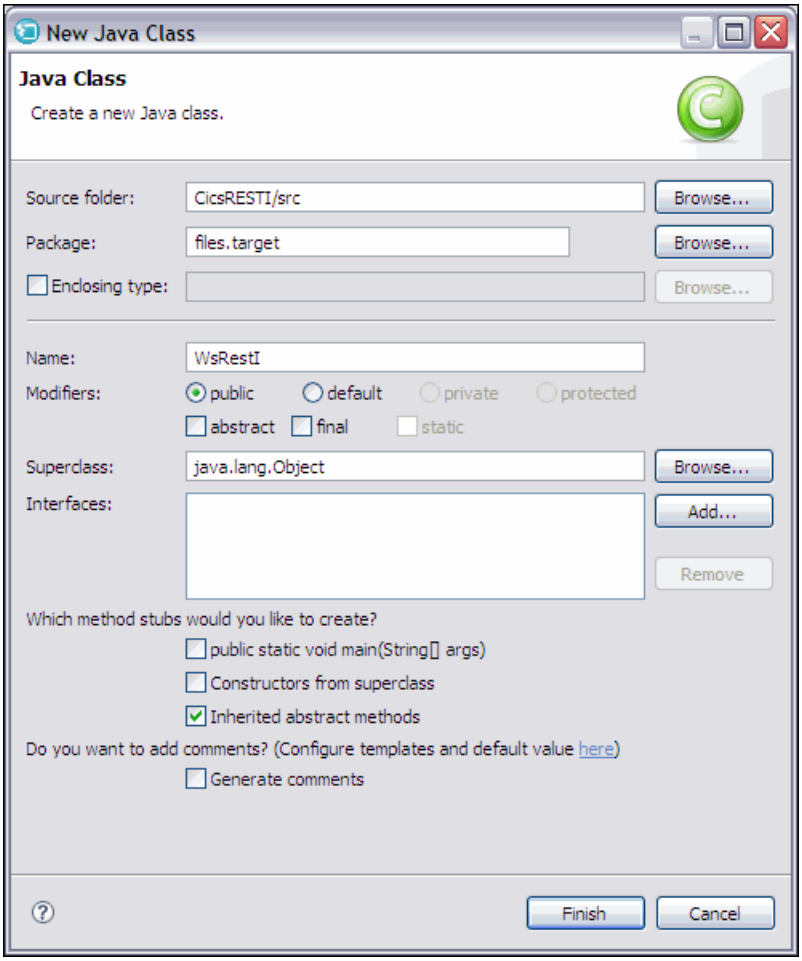


Figure 7-9 Creating a new Java class for REST

6. Click **Finish.**

   You will now see a Java source editor opened with a basic Java class definition. We now add a new method to call a Web service with appropriate parameters and get the response. To do so we create a Web service proxy object and call a Web service operation through that. We need to add the following code to the Java class `WsRestI`.

   `custproxy` Is a new object declared for Web service proxy calls.

   `callWS` Is a new method added to call Web service operation and get the response back. This method needs `custNo (int)` as an input and returns customer details as `com.CUSTINQO.www.DFHCOMMAREA object`.

7. Complete the `WsRestI` class source code so that it looks as shown in Example 7-1 on page 134.

Example 7-1 wsRestI source code

```
package files.target;

public class WsRestI {

   CUSTINQPortTypeProxy custproxy = new CUSTINQPortTypeProxy();

   public com.CUSTINQO.www.DFHCOMMAREA callWS(int custNo) {

      com.CUSTINQI.www.DFHCOMMAREA custCOMM = new
com.CUSTINQI.www.DFHCOMMAREA();

      custCOMM.setCustNo(custNo);

      return custproxy.CUSTINQOperation(custCOMM);

   }

}
```

You will probably now see an error message in the editor described as `unhandled exception type RemoteException`. Since we are trying to call a remote interface, we need to implement `RemoteException` for this class.

8. Just click **quick fix** and select **Add throws declaration** from the suggestions, as shown in Figure 7-10.

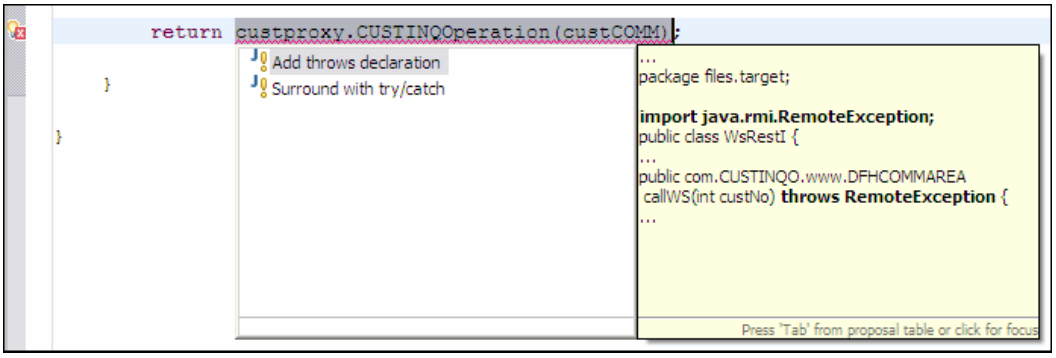


Figure 7-10 Adding a throws exception to the source code

This will change the class declaration with a `throws` statement, as shown in Example 7-2.

Example 7-2 Changed class declaration

```
public com.CUSTINQO.www.DFHCOMMAREA callWS(int custNo) throws RemoteException
{
```

9. Press Ctrl+s to save the source file.

7.4.4 Exposing the WsRestI class as a REST service

Now that we have the Web Services class with a call to a Web Services operation, we can expose it as a REST service.

1. Right-click the **WsRestI.java** file in the Project Explorer view and select **Services** → **Expose RPC Adapter service** from the context menu.

   You will see a pop-up dialog with the title Expose RPC Adapter Service. Select the method you want to expose as a REST service, as shown in Figure 7-11.

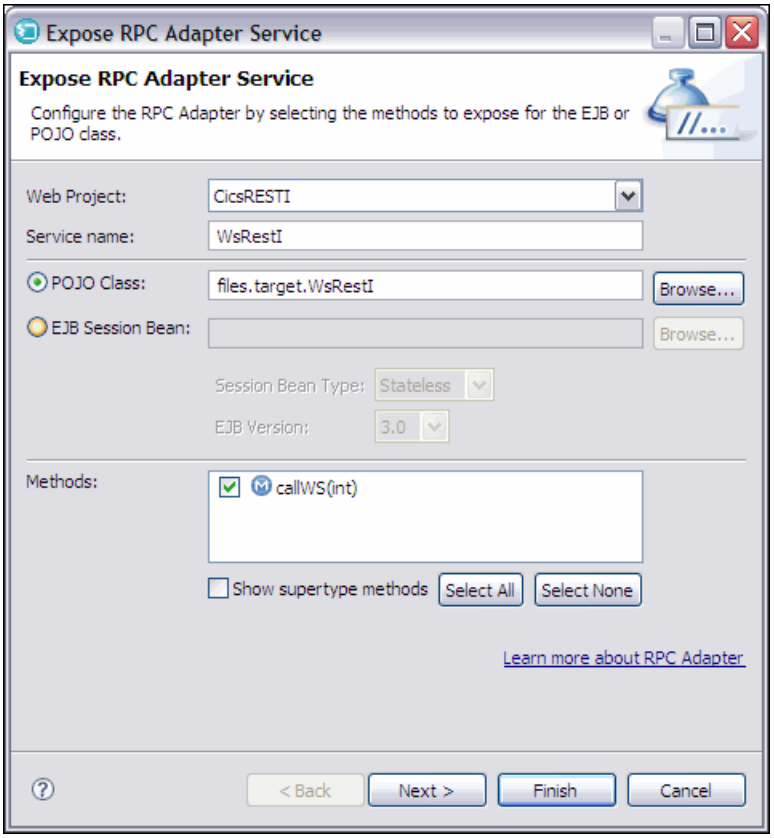


Figure 7-11 Selecting the method as a REST service

2. Click **Next >**.
3. You can configure the HTTP request type as GET or POST and you can see the address to call the method remotely over HTTP.Use the default, so the HTTP Method is GET. The method to be called is `callWS` with an integer parameter (Customer ID) and the address is `/CicsRESTI/RPCAdapter/httprpc/WsRestI`. See Figure 7-12 on page 136.

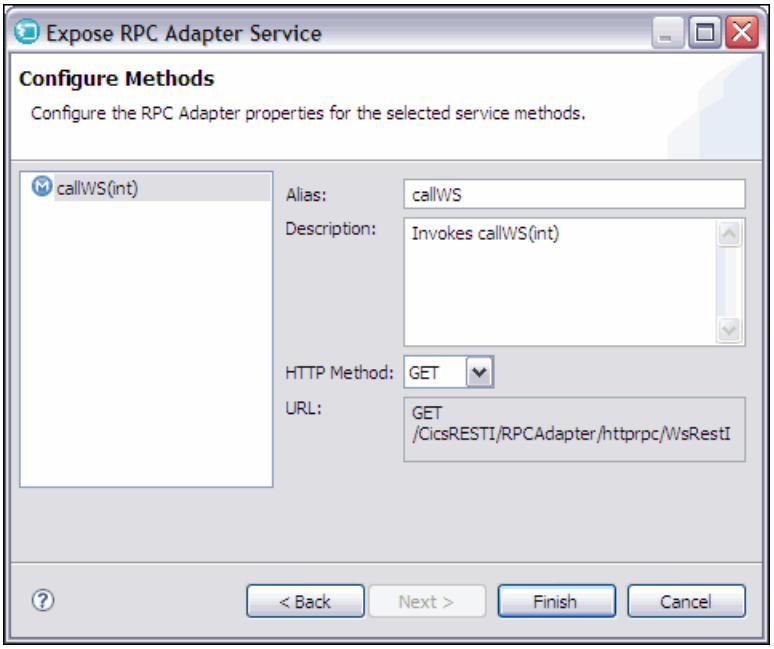


Figure 7-12 Configuring the methods

4. Click **Finish.**
5. Expand the Services folder in the Project Explorer and further expand to **WsRestI → callWS(int)** and you can see that the class and method are now exposed as a REST service.
6. You can also explore the RpcAdapterConfig.xml file recently added to the project. Double-click it to open the RPC Adapter configuration editor. See Figure 7-13.

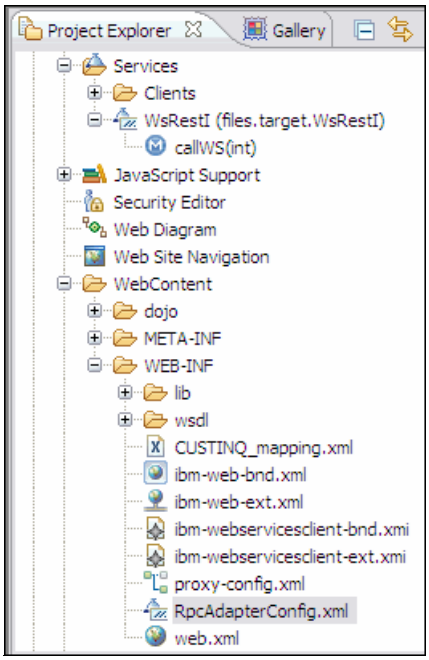


Figure 7-13 Selecting the RpcAdapterConfig.xml file

This opens an editor window, called the RPC Adapter Configuration Editor, as shown in Figure 7-14.

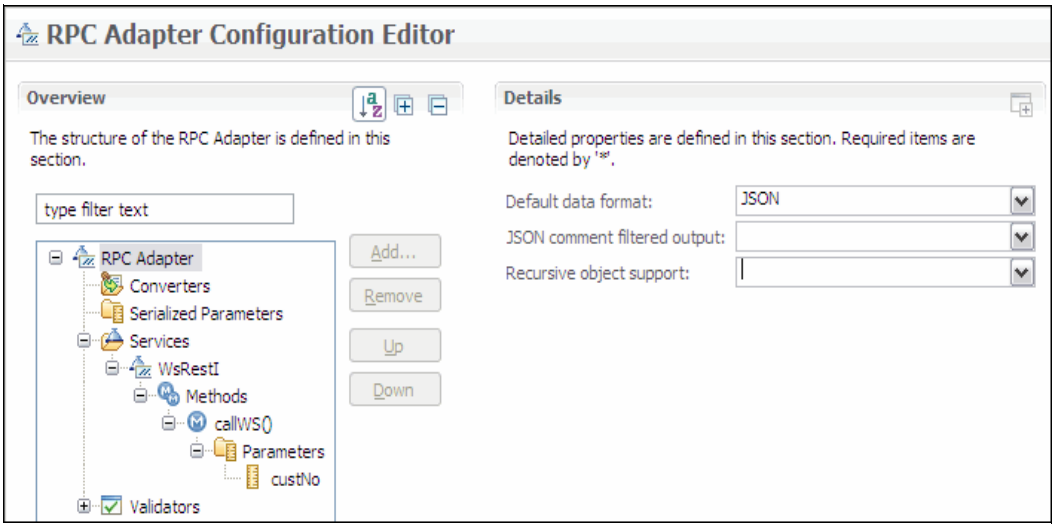


Figure 7-14 RPC Adapter Configuration Editor

7. You can also switch to the Source view of the RPC Adapter Configuration Editor. The configuration information for the REST service can be found between the `<services>` and `</services>` tags, as shown in Example 7-3.

Example 7-3 RpcAdapterConfig.xml source

```
<services>
      <pojo>
         <name>WsRestI</name>
         <implementation>files.target.WsRestI</implementation>
         <methods filter="whitelisting">
            <method>
               <name>callWS</name>
               <alias>callWS</alias>
               <description>Invokes callWS(int)</description>
               <http-method>GET</http-method>
               <parameters>
                  <parameter>
                     <name>custNo</name>
                     <type>int</type>
                     <description></description>
                  </parameter>
               </parameters>
            </method>
         </methods>
      </pojo>
   </services>
```

The default data format is set to JSON. It can be XML as well. But JSON has several advantages over XML in developing Web 2.0-based applications.

Note: JavaScript Object Notation (JSON) is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

The JSON format was specified in RFC 4627 by Douglas Crockford. The official Internet media type for JSON is application/json. The JSON file extension is.json. The JSON format is often used for transmitting structured data over a network connection in a process called serialization. It is primarily used in Ajax Web application programming, where it serves as an alternative to the traditional use of the XML format.

Although JSON is based on a subset of the JavaScript programming language (specifically, Standard ECMA-262 3rd Edition—December 1999) and is commonly used with that language, it is considered to be a language-independent data format. Code for parsing and generating JSON data is readily available for a large variety of programming languages. The JSON Web site:

`http://json.org/`

provides a comprehensive listing of existing JSON bindings, organized by programming language.

8. To publish the REST service to WebSphere Application Server, you can right-click **Services** → **WsRestI** → **callWS (int)** in the Project Explorer and select **Run As** → **Run on Server** from the context menu.
9. This will pop up the Run On Server dialog and you can simply click **Finish.** The REST service will be published to WebSphere Application Server. It will also open up a browser and ask to save the file callWS. Save this file and open it with Wordpad.

   The contents of the text file will be an error message in JSON format, as shown in Example 7-4.

   Example 7-4 JSON error

```
{"error":{"serviceName":"WsRestI","message":"CWRPC0008E: The parameters in the
request do not match the parameters that are defined for the callWS
method.","parameterValue":null,"parameterIndex":null,"name":"JSONRPCError","met
hodName":"callWS","stackTrace":null,"code":"CWRPC0008E"}}
```

   This error has appeared because we have not passed the parameter required to get the response.

   To test the REST service, you can also use the *Firefox Poster* plugin. More information about Poster can be found at:

   `https://addons.mozilla.org/en-US/firefox/addon/2691`

10. We will try to call the REST service from the Poster user interface, as shown in Figure 7-15 on page 139.
    - Update the address to `http://localhost:9081/CicsRESTI/RPCAdapter/httprpc/WsRestI/callWS`
    - Action type is GET.
    - Add a parameter with `custNo` as name and 2 as value. And click **Submit.**

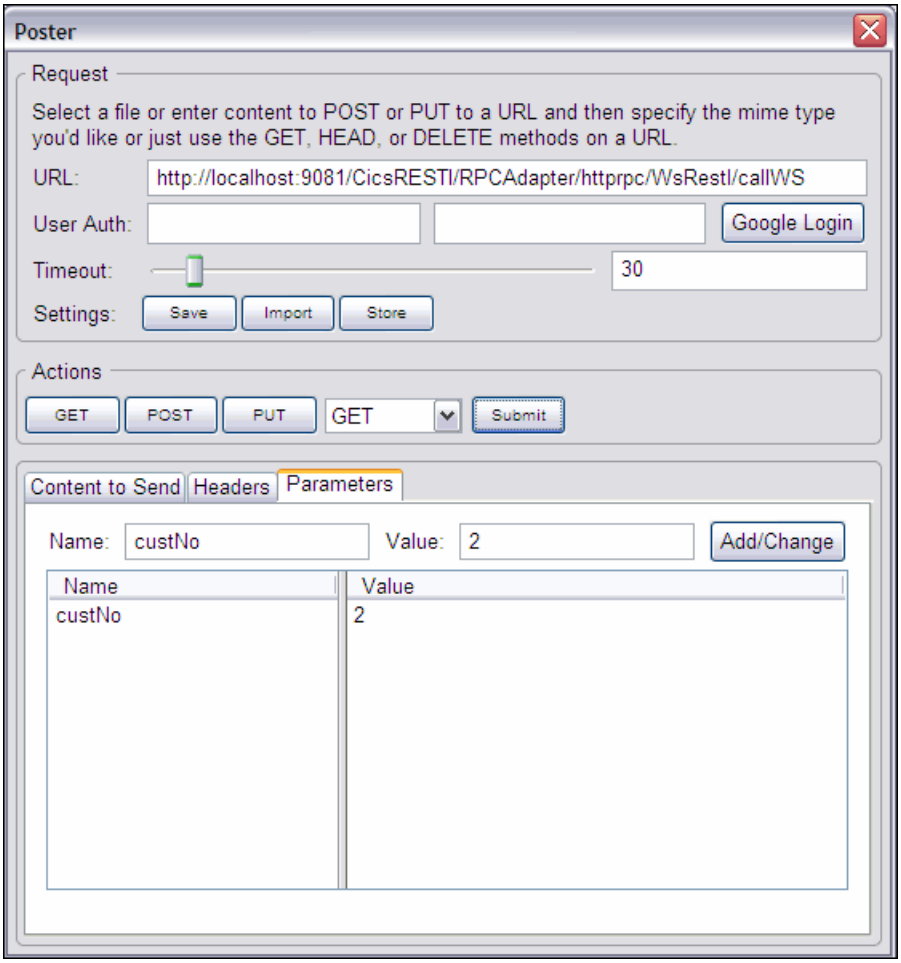


Figure 7-15 Testing a REST service with Firefox Poster

You will see a JSON response returned from the REST service. It includes customer details returned from the Web service. See Figure 7-16 on page 140.

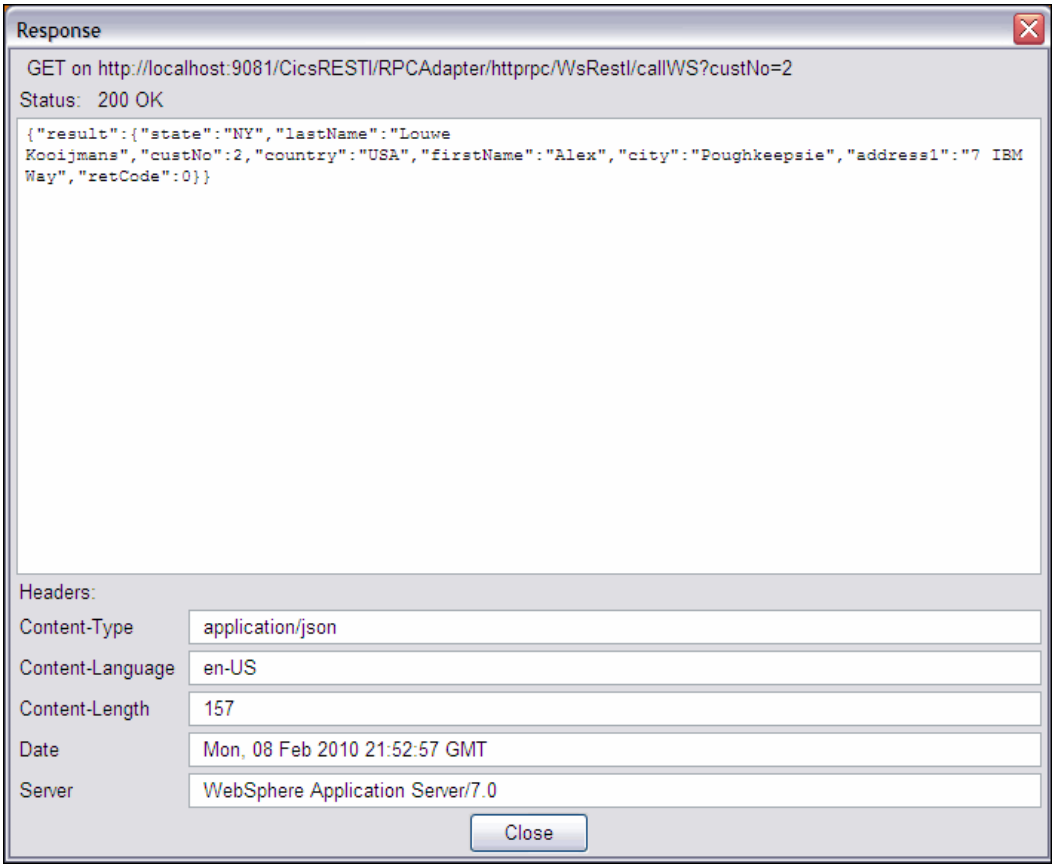


Figure 7-16 Response from the REST service in Firefox Poster

You have now successfully deployed a REST service in WebSphere Application Server that calls a Web service in CICS.

7.4.5 Adding a Web client for iPhone calling the REST service

To add a REST Web client we now create another Dynamic Web Project in the workbench. One HTML page from this project will call the REST service.

1. To create a new Dynamic Web Project as discussed earlier, select **File** → **New** → **Dynamic Web Project** from the RDz workbench toolbar.
2. Update the following in the pop-up dialog (Figure 7-17 on page 141):

   Project Name CicsRESTClient

   Target Runtime WebSphere Application Server 7.0

   Dynamic Web Module Version 2.5

   Select the check box **Add project to an EAR** and specify the EAR file name as `CicsRESTIEAR`.
3. We need to change the default configuration for WebSphere Application Server, so click **Modify** adjacent to the Configuration text field.
4. This will pop up the Project Facets dialog. Select **Web 2.0** and **JavaScript Toolkit** from the list. Click **OK.** You will now return to the New Dynamic Web Project dialog and the configuration text box will be updated with **<custom>.** By selecting **Web 2.0** from the list you are adding support for the Dojo toolkit-specific libraries in our project.

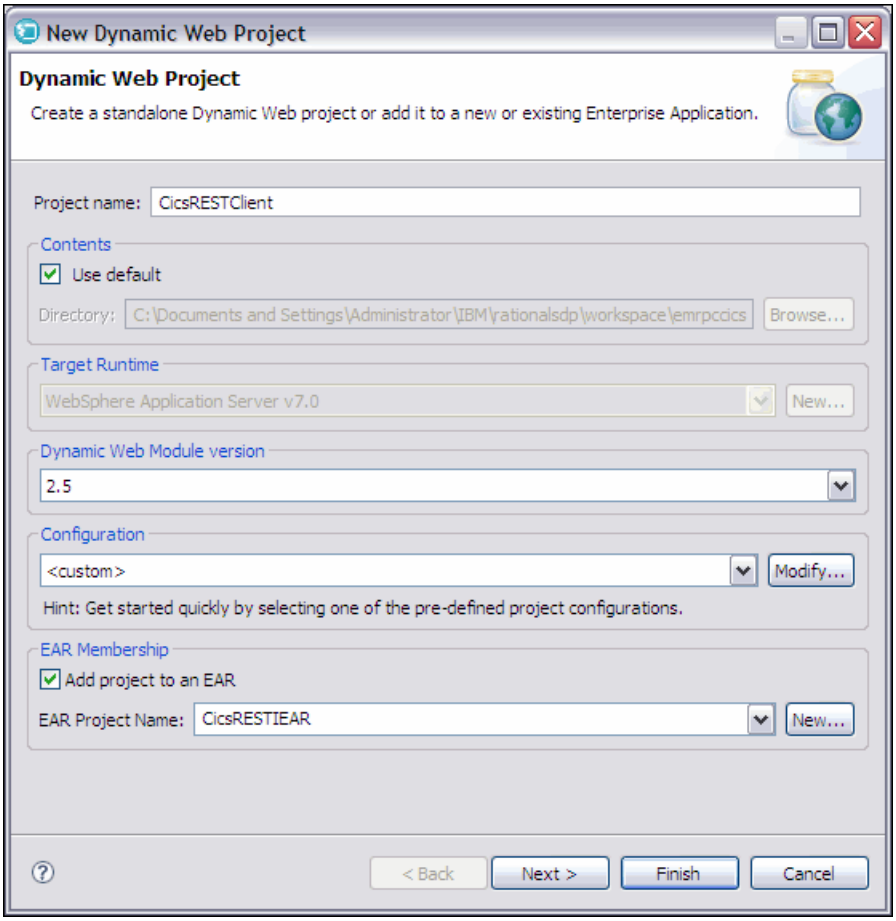


Figure 7-17 Creating a new Dynamic Web Project

5. Click **Finish.**
6. When you have added a new project, the workbench's Project Explorer should look as in Figure 7-18.

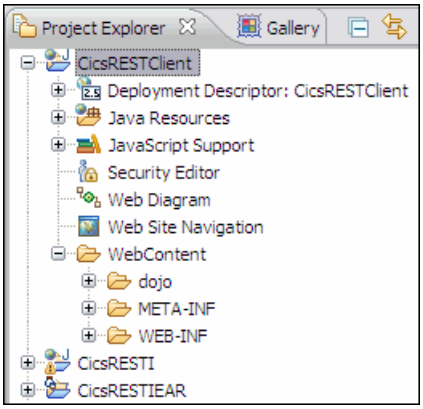


Figure 7-18 Project Explorer with new Dynamic Web Project added

7. To add a new HTML page, right-click the **CicsRESTClient** project in the Project Explorer and select **New** → **Web Page** from the context menu.
8. Enter `RestClient` as **File Name** and `HTML/XHTML` as **Template**, as shown in Figure 7-19.

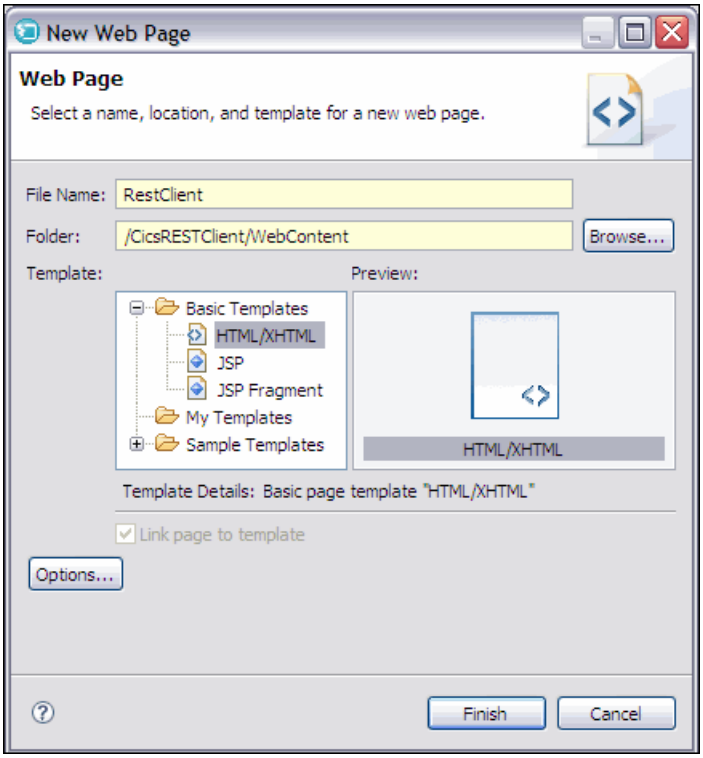


Figure 7-19 Web page details

9. Click **Finish.** You will see the `RestClient.html` file opened in the HTML editor in the workbench.

   With the help of the Palette view you can drag and drop HTML or Dojo components and the Source view of the HTML editor will help you in adding JavaScript to the code as well.

   The Properties view can help you quickly define the setting of HTML or Dojo components such as stylesheet definitions, width, colors, and so on. You can also create a new stylesheet definition and apply it from the Properties view.

   We have created a basic view to enter the customer ID and view customer details as shown in Figure 7-20 on page 143.

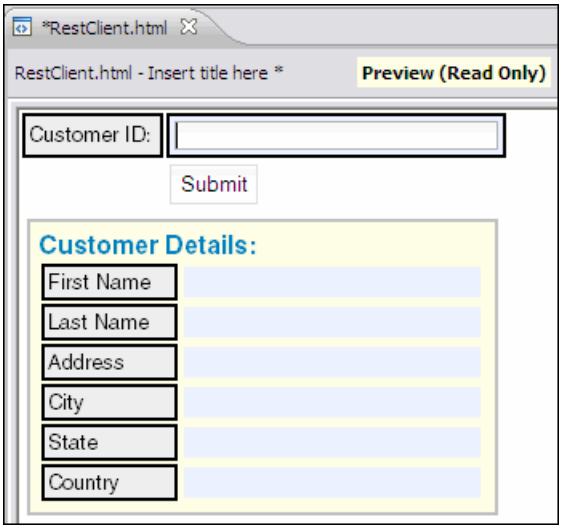


Figure 7-20 Basic HTML page to enter Customer ID and see Customer details

10. You will also need to add the following meta tag in the title section to fit on the page on an iPhone browser. as shown in Example 7-5.

Example 7-5 Meta tag in HTML page for page size

```
<meta name="viewport" content="width=320,user-scalable=no">
```

While adding the line, you may get a pop-up message as shown in Figure 7-21. Just click **Yes.**

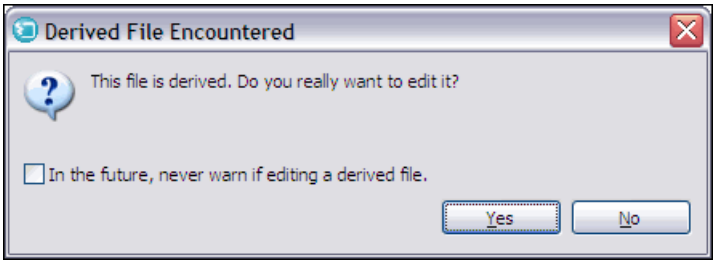


Figure 7-21 Derived file modification confirmation message

The following JavaScript may be needed to display the HTML page as desired:

– In case you want to remove the address bar from the browser and want the application to look like a native Apple iPhone application, you can add the JavaScript shown in Example 7-6 on page 144 to your HTML page.

Example 7-6 JavaScript to hide the address bar in the HTML page

```
<script type="application/x-javascript">
    if (navigator.userAgent.indexOf('iPhone') != -1)
    {
        addEventListener("load", function()
        {
        setTimeout(hideURLbar, 0);
        }, false);
    }
    function hideURLbar()
    {
        window.scrollTo(0, 1);
    }
</script>
```

– If you want to add a custom icon for your application when a user bookmarks the application to the home panel, you can do so by adding the JavaScript shown in Example 7-7 to the HTML page. We have an icon file located in the icons folder with the name `Finder.png` that will be used as an icon for the application bookmark.

Example 7-7 JavaScript to add a custom icon

```
<link rel="apple-touch-icon" href="icons/Finder.png"/>
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style"
content="black"/>
```

11. We have also created a couple of style classes to be referred from HTML or Dojo components. These are tablecell, tablecellresult and titlelabel. See Example 7-8.

Example 7-8 Style classes added

```
.tablecell {
   font-style: normal;
   width: 80px;
   background-color: #efefef;
   color: black;
   border-style: solid;
   border-width: thin;
   font-size: 10pt;
   font-weight: 400;
   border-color: black;
   padding: 2px
}

.tablecellresult {
   font-style: normal;
   background-color: #ecf1ff;
   color: black;
   border-style: solid;
   border-width: thin;
   font-size: 10pt;
   font-weight: 400;
   border-color: black;
```

```
   padding: 2px;
   width: 200px
}

.titlelabel {
   font-style: normal;
   background-color: #fefde7;
   margin: 5px;
   color: #0080c0;
   border-style: solid;
   border-width: thin;
   font-size: 12pt;
   font-weight: bold;
   border-color: silver;
   padding: 5px;
   width: 280px
}</style>
```

The body section of the ResultClient.html page is shown in Example 7-9.

Dojo or HTML components added in the body section refer to stylesheets classes defined earlier. Moreover, it also uses Dojo toolkit's look and feel. Formatting of components has been done with the help of Table tags. Every Dojo or HTML component has been assigned a unique ID. See Example 7-9.

Example 7-9 Dojo / HTML components added

```
<body class="nihilo">
<div dojotype="dijit.form.Form" id="Form">
<table>
<tr>
<td class="tablecell">Customer ID: </td><td class="tablecellresult"><input
dojotype="dijit.form.TextBox" id="custid"></td></tr><tr>
<td></td><td><button dojotype="dijit.form.Button"
onclick="read()">Submit</button></td></tr>
</table>
</div>

<div id="resultform" class="titlelabel">Customer Details:<BR>
<table>
<tr>
<td class="tablecell">First Name</td><td class="tablecellresult"> <div id="firstname"
></div></td>
</tr><tr>
<td class="tablecell">Last Name</td><td class="tablecellresult"><div id="lastname"></td>
</tr><tr>
<td class="tablecell">Address</td><td class="tablecellresult"> <div id="address"></td>
</tr><tr>
<td class="tablecell">City</td><td class="tablecellresult"> <div id="city"></td>
</tr><tr>
<td class="tablecell">State</td><td class="tablecellresult"> <div id="state"></td>
</tr><tr>
<td class="tablecell">Country</td><td class="tablecellresult"><div id="country"></td>
</tr>
</table>
</div>
</body>
```

12. Now we add JavaScript code to call the REST service and display results in Dojo or HTML components.

    We discuss different sections of JavaScript code now. The first section is required by the Dojo toolkit to load Dojo's HTML components in the browser, and it adds the `dojo.require` statement for every component that is required on the HTML page, as shown in Example 7-10.

Example 7-10 JavaScript to load Dojo's HTML components

```
<script type="text/javascript" src="dojo/dojo/dojo.js"
   djConfig="isDebug: false, parseOnLoad: true"></script>
<script type="text/javascript">
dojo.require("dojo.parser");
dojo.require("dijit.form.Form");
dojo.require("dijit.form.TextBox");
dojo.require("dijit.form.Button");
```

In the next section of JavaScript we create an XMLHTTP request object that is used to send a request to REST service and process the response.

When **Submit** is clicked, the `read()` function is called. It reads the value inserted in the **custid** text field and calls the CreateXMLHttpRequest() function, which in turn creates an XMLHttpRequest object and sends a GET request to the REST address as shown in Example 7-11.

Example 7-11 JavaScript for creating an XMLHTTP request object

```
var xmlHttp;

function createXMLHttpRequest() {
   if(window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   else {
      xmlHttp = new XMLHttpRequest();
   }
}

function read() {
   var custno = document.getElementById('custid').value;

   createXMLHttpRequest();
   xmlHttp.onreadystatechange = handleStateChange;

xmlHttp.open("GET","http://localhost:9081/CicsRESTI/RPCAdapter/httprpc/WsRestI/
callWS?custno="+custno,true);
   xmlHttp.send(null);
}
```

Once the response is received from the REST service, the response object is forwarded to the `handleStateChange()` function for further processing.

The `handleStateChange()` function processes the response and verifies the validity of the response returned, as shown in Example 7-12 on page 147.

Example 7-12 JavaScript to validate the response

```
function handleStateChange()
{
   if (xmlHttp.readyState == 4 ) {
      if (xmlHttp.status == 200) {
        var custDetails = eval("("+xmlHttp.responseText+")");
        if(custDetails != null)
           updateResultGrid(custDetails);
         }
      }
   }
```

After successful verification of the response object the `handleStateChange()` method creates a `custDetails` object and forwards it to `updateResultGrid()` with `custDetails` as a parameter.

The JavaScript method `updateResultGrid()` updates the response values on the HTML page to display the page to the user, as shown in Example 7-13.

Example 7-13 JavaScript for updating the response values

```
function updateResultGrid(custDetails)
{
document.getElementById('firstname').innerHTML=custDetails.result.firstName;
document.getElementById('lastname').innerHTML=custDetails.result.lastName;
document.getElementById('address').innerHTML=custDetails.result.address1;
document.getElementById('city').innerHTML=custDetails.result.city;
document.getElementById('state').innerHTML=custDetails.result.state;
document.getElementById('country').innerHTML=custDetails.result.country;
}
```

7.4.6 Testing the client application

To test the client you can right-click the **RestClient.html** page and select **Run As** → **Run on Server** from the context menu. It will publish the REST service project as well as the client project at WebSphere Application Server. In our case the address is:

`http://9.57.139.23:9081//CicsRESTClient/RestClient.html`

You should use your own IP address and port number for testing in your environment.

Once we point the Apple iPhone browser to the address of RestClient.html, we can observe the page on the Apple iPhone, as shown in Figure 7-22 on page 148.

Figure 7-22 Apple iPhone accessing a REST service

You can insert the customer ID as 002 for testing purposes and press **Submit.** The response from the Web service or REST service will be displayed on the panel, as shown in Figure 7-23.

Figure 7-23 Response on the iPhone

We have successfully tested a REST service client calling a CICS Web service through a REST interface.

7.4.7 Deployment to WebSphere Application Server

You can also deploy the JEE application to WebSphere Application Server on z/OS. Refer to Appendix C, “Deployment to WebSphere Application Server on z/OS” on page 197.

8

Accessing IBM Tivoli Monitoring from smartphones

An interesting use of smartphones could be to receive events or alerts whenever there is a situation on z/OS that needs attention. The IBM Tivoli Monitoring (ITM) SOAP interface enables you to integrate the Web world with ITM so that you can exchange information between ITM on System z and a smartphone.

In this chapter, we discuss how to build a client application for a smartphone that does exactly that. To learn more about the ITM architecture and its SOAP interface, refer to Appendix B, “IBM Tivoli Monitoring and the SOAP interface” on page 177.

© Copyright IBM Corp. 2010. All rights reserved.

8.1 Introduction to IBM Tivoli Monitoring

IBM Tivoli Monitoring (ITM) is a monitoring and reporting framework that enables monitoring agents on a wide range of platforms to collect, monitor, automate, and display performance and availability metrics from a wide variety of products.

On the z/OS platform, ITM has monitoring agents for components such as the operating system itself, the storage (disk and tape) subsystem, CICS, IMS, DB2, MQ, and WebSphere etc. As a result ITM has the potential to be a huge source of performance monitoring and availability data.

ITM also offers a SOAP interface that, if configured, can be used to retrieve monitoring data in real time from any monitoring agent that is connected to the ITM framework, subject of course to any security restrictions that might be implemented by the user regarding which users can access which applications, and so on. Using the SOAP interface in conjunction with some thoughtful programming can create a very usable Web-based interface to small subsets of the available monitoring data. This frees you from the need to be in front of a computer panel in order to see what is happening in critical areas of your z/OS environment.

8.2 Architecture overview

You could create a smartphone application that communicates directly with the ITM SOAP server. However, such a two-tier implementation may easily lead to a situation where you are sending too much data to the client with obvious performance implications. Also, the ITM server does not provide all the typical Web server functionality that you find in, for example, WebSphere Application Server. Therefore, a three-tier architecture would be much more appropriate with the benefit of leveraging all the WebSphere functionality with respect to serving Web clients, including smartphones. You can use WebSphere Application Server to implement additional server-side code to support mobile devices efficiently and perform operations on data before sending it on to the mobile client device. A three-tier architecture is also far more secure.

The client in this three-tier scenario can actually be built using different technologies. Again, Enterprise Generation Language (EGL) Rich UI is a perfect option, which is provided in both the Rational Business Developer (RBD) and specific versions of the Rational Developer for System z (RDz) products. However, this time we have chosen to use AJAX and Dojo to develop the client.

The scenario in this chapter is shown in Figure 8-1 on page 153.

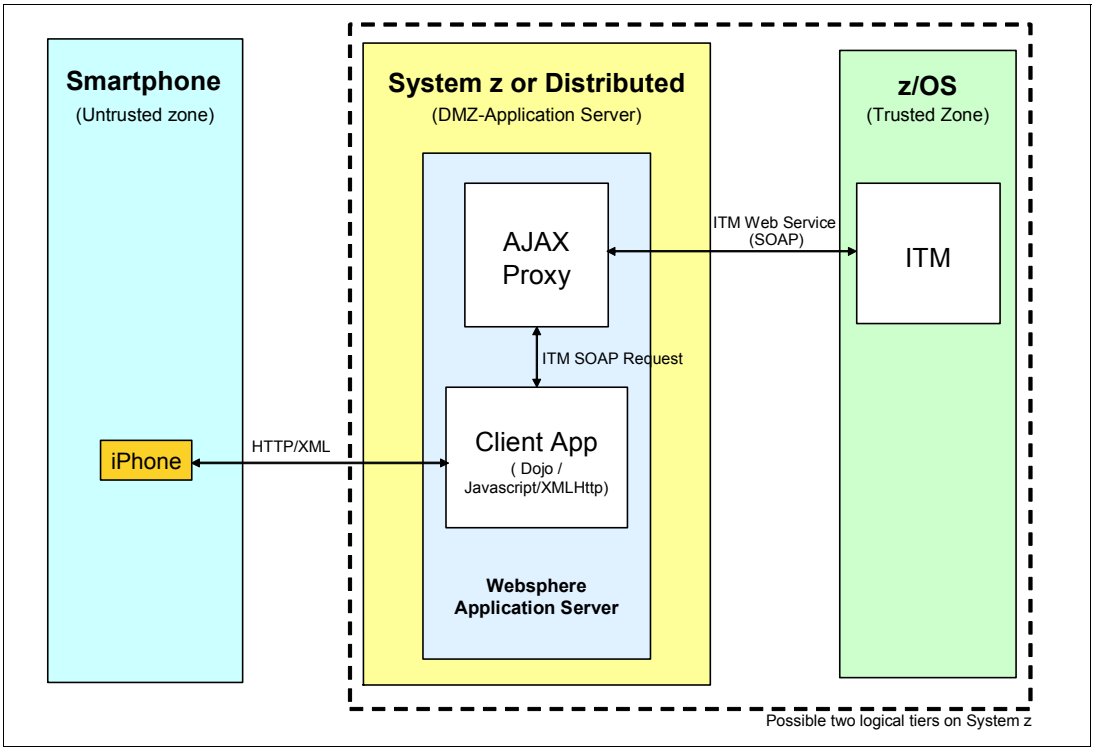


Figure 8-1 Solution architecture - Accessing ITM from a smartphone

Of course, there are many ways to organize the components of your enterprise management environment, and this shows a highly simplified view of an implementation. Ultimately, though, it boils down to the server providing both Web Services to the mobile clients and providing a proxy and additional processing for the data provided by the ITM SOAP interface, which typically resides on the ITM hub.

8.3 Some key considerations

As per the ITM architecture, if a monitoring agent collects data then, in theory, you can view that information on a smartphone. However, the sheer volume of data that comes out of z/OS systems can easily overwhelm a device such as a smartphone. What you really have to decide in this arena is why you need to see certain data and what the most critical aspect is.

For example, you may want to see an overview of the CPU usage of all address spaces on a z//OS system. But why do you need to see them all? Possibly, in the event of, say, an address space consuming a lot of CPU, you may want to see the high CPU users. In that case you might want to see the top ten CPU users on the system. There is little to be gained in this situation from seeing more data.

The point is that the challenge of viewing performance monitoring data on a smartphone is actually about reducing the volume of data to a level that does not overwhelm the device or the user, whilst still enabling you to obtain useful information that enables you to take some action to a situation, based on the knowledge that the data gives you.

You could, with the appropriate programming and infrastructure, initiate a limited set of predefined actions, for example restarting a failed address space or cancelling certain jobs, directly from your smartphone. Again, it is not intended for everyday and volume use, but it does mean you can support your system intelligently while away from your workstation.

8.4 Software used

For this scenario we used the following software:

- IBM Tivoli Monitoring
- Rational Developer for System z with Java 7.6
- WebSphere Application Server 7.0
- A browser, such as Safari, Microsoft Internet Explorer, or Firefox

8.5 Back-end requirements

An additional requirement for the ITM back end is to have the SOAP connectivity interface of IBM Tivoli Monitoring system available.

8.6 Step-by-step instructions

In this section we discuss the detailed instructions to develop and test a rich client application with interaction with the ITM server's SOAP interface.

8.6.1 Sample code

The code developed for this scenario is available for download as an EAR file as well as a Project Interchange File. Refer to Appendix D, "Additional material" on page 203 for instructions.

8.6.2 Creating a Dynamic Web Project for the ITM client

The first thing to do is to create a new Dynamic Web Project in RDz. We followed these steps:

1. We launched Rational Developer for System z (RDz) and specified the workspace name. (The workspace is a folder where all the artifacts related to the project will be stored.)
2. Once the workbench was started, we switched to the Web perspective. This can be done by selecting **Window** → **Open Perspective** → **Other ...** and then selecting **Web** from the pop-up dialog.
3. We selected **File** → **New** → **Dynamic Web Project** from the RDz workbench's menu bar, and updated the following fields in the pop-up New Dynamic Web Project dialog, as shown in Figure 8-2:

   Project name: The name of the project. We chose ITMClient.

   Target Runtime: WebSphere Application Server v7.0.

   Dynamic Web Module version: 2.5

   Configuration: We clicked **Modify** adjacent to the configuration drop-down list. The Project Facets pop-up window appeared and we selected the check box for **web 2.0** in the Project Facets list. We clicked **OK.** The Configuration changed to **<custom>.**

   We selected the check box for **Add project to an EAR.** The suggested EAR file name is automatically set based on what you entered in "Project name:". In our case the name of the EAR file was `ITMClientEAR`.

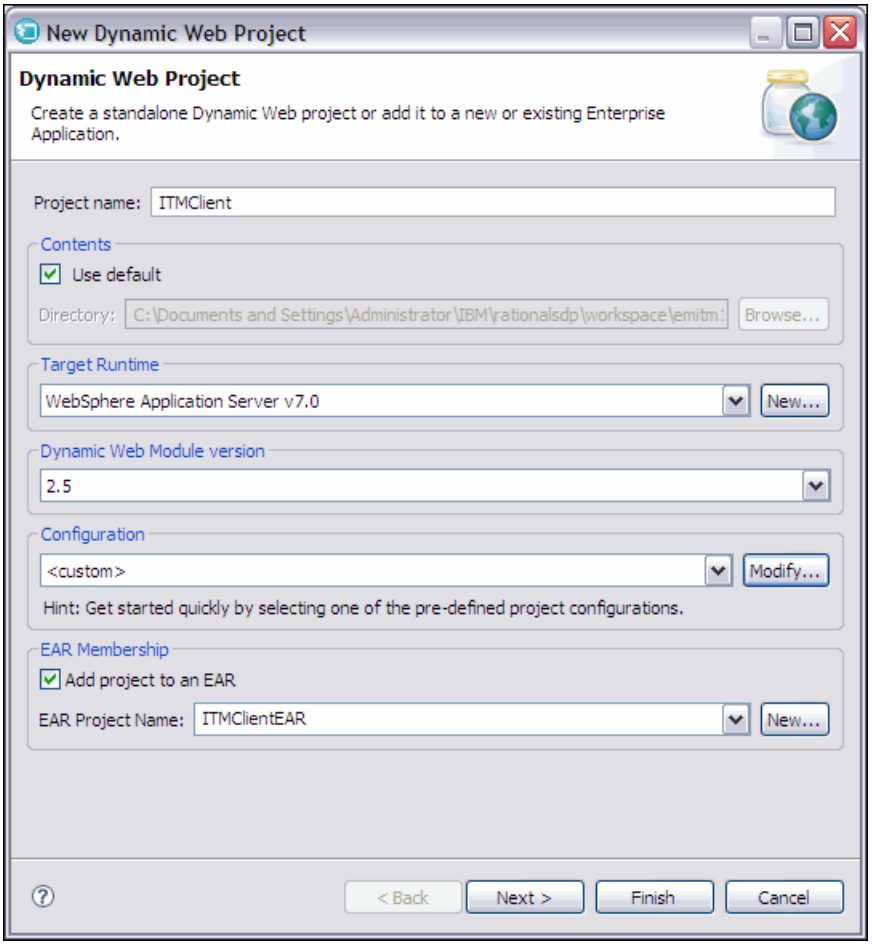


Figure 8-2 Entering details for the Dynamic Web Project

4. We clicked **Finish.**

   A complete project hierarchy can now be seen in the Project Explorer view. We now added a new Client.html page to the Web project ITMClient. To do this, we right-clicked the ITMClient project in the Project Explorer and selected **New** → **Html** from the pop-up context menu.

5. We inserted the file name `sample` and clicked **Finish.** See Figure 8-3 on page 156.

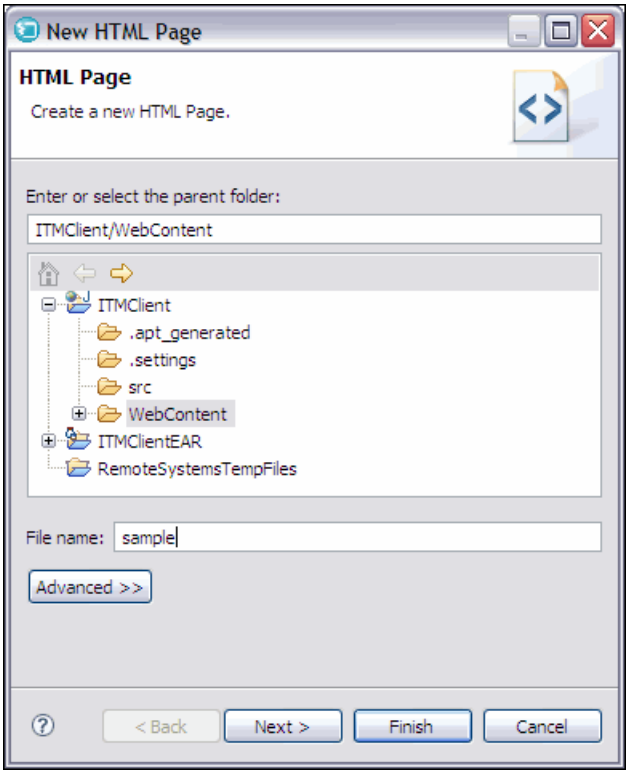


Figure 8-3 Adding an HTML page

The ITM client project structure in the Project Explorer looked similar to that shown in Figure 8-4.

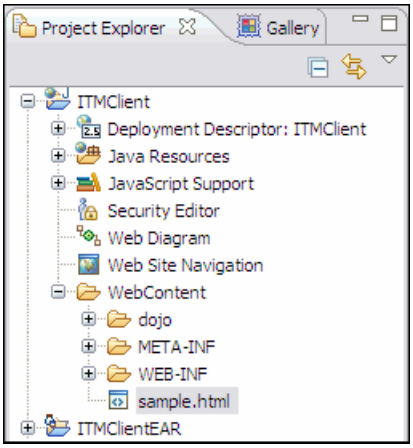


Figure 8-4 Project Explorer

Now that we have a Dynamic Web Project and HTML page in the project, we can start designing a user interface and add JavaScript to call the ITM SOAP service.

8.6.3 Designing a user interface to display information

In our sample we used the Dojo Toolkit (`www.dojotoolkit.org`), a JavaScript toolkit that can be used to create rich user interfaces for Apple iPhone. The Dojo Toolkit is bundled with RDz with Java.

You can use the **Dojo Form Widgets** tab of the **Palette** to drag and drop Dojo widgets to the sample.html page. To get started we added text boxes and labels for user name, password, and also a button to submit the request, as shown in Figure 8-5.

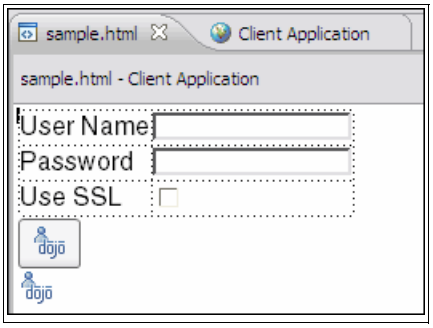


Figure 8-5 Rich user interface design with Dojo

We also added a grid widget in the list where all the systems and their state will be displayed once we get a response from the ITM server. See Example 8-1.

Example 8-1 Source view for Dojo widgets added to the client page

```
<script type="text/javascript">
dojo.require("dijit.form.CheckBox");
dojo.require("dijit.form.Form");
dojo.require("dijit.form.TextBox");
dojo.require("dijit.form.Button");
dojo.require("dojo.parser");
dojo.require("dojox.grid.compat.Grid");
dojo.require("dojox.grid.Grid");
dojo.require("dojox.grid.compat._data.model");
</script>
```

The **Outline** view can also help you here to quickly visualize and understand the sample.html page **body** section, as shown in Figure 8-6 on page 158.

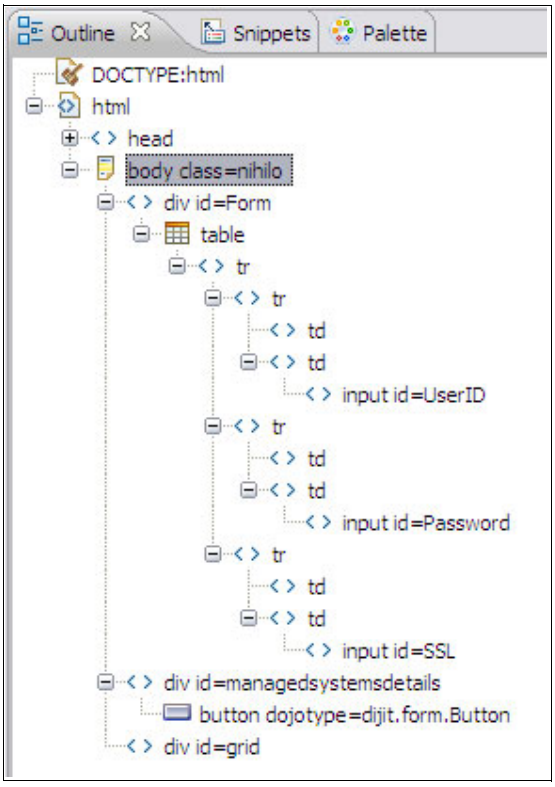


Figure 8-6 Body outline of the sample.html page

8.6.4 Adding JavaScript to interact with the ITM SOAP server

Now that we have our basic user interface structure ready, it is time to make it interact with the ITM server. To do so we added a couple of JavaScript functions to connect to the ITM Server, send a request and then expect a response. You can see a list of these functions in the Outline view for the sample.html page, as shown in Figure 8-7 on page 159.

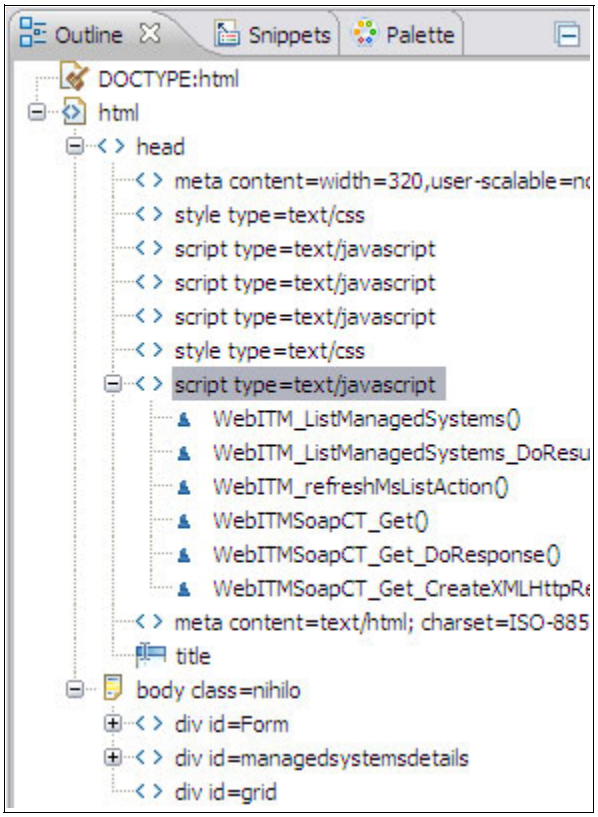


Figure 8-7 - JavaScript functions added in the sample page

You can also add all these Javascript functions in any common js file and refer to the js file from the sample.html page by using:

```
<script type="text/javascript" src="<path of js file>"></script>
```

We will now discuss the functionality implemented in these JavaScript functions.

WebITMSoapCT_Get_CreateXMLHttpRequest

This function creates an `XMLHttpRequest` object and returns it as an `xmlHttp` variable.

WebITM_ListManagedSystems

This function prepares a query (in XML Schema format) and forwards it to the `WebITMSoapCT_Get` function for preparing a final Web Services request to be sent to the ITM server. It also sets the `WebITM_ListManagedSystems_DoResults` as a call-back function for processing the results received from the ITM server.

WebITM_ListManagedSystems_DoResults

This function processes the XML response received from the ITM server, extracts the results, and displays them on the Dojo grid control added to the browser.

WebITM_refreshMsListAction

This function picks up values such as user ID and password and assigns them to different variables to be used later in a script for processing the request. It also calls the `WebITM_ListManagedSystems` function for further SOAP request processing.

WebITMSoapCT_Get

This function prepares an HTTP request to be sent to the ITM server (routed through an Ajax Proxy) with parameters such as user ID, password, and the XML schema prepared in the `WebITM_ListManagedSystems` function. It also sets header parameters for the `XMLHttpRequest` object received from the `WebITMSoapCT_Get_CreateXMLHttpRequest` function. And finally it specifies the `WebITMSoapCT_Get_DoResponse` function as a call-back function to handle the response received from the ITM server.

WebITMSoapCT_Get_DoResponse

This is the response handler function for the `WebITMSoapCT_Get` function.

You can find the source of these JavaScript functions in the additional material section of this book, and you can customize the source as per your infrastructure requirements.

Once we have added these JavaScript functions to the page, we can call the `WebITM_refreshMsListAction` function with a click of the button by adding the code snippet, as shown in Example 8-2.

Example 8-2 Calling a JavaScript function when clicking a button

```
<button dojotype="dijit.form.Button"
   onclick="WebITM_refreshMsListAction()">Managed Systems Details</button>
```

JavaScript to optimize the HTML page for the smartphone

You also need to add the following meta tag in the title section of the page, as shown in Example 8-3.

Example 8-3 HTML - Meta tag to be added in the page

```
<meta name="viewport" content="width=320,user-scalable=no">
```

The following JavaScript might be needed to display the HTML page as desired:

- In case you want to remove the address bar from the browser and want the application to look like a native Apple iPhone application, you can add the JavaScript shown in Example 8-4 to your HTML page.

  Example 8-4 JavaScript to hide the address bar in the HTML page

  ```
  <script type="application/x-javascript">
      if (navigator.userAgent.indexOf('iPhone') != -1)
      {
          addEventListener("load", function()
          {
          setTimeout(hideURLbar, 0);
          }, false);
      }
      function hideURLbar()
      {
          window.scrollTo(0, 1);
      }
  </script>
  ```

- If you want to add a custom icon for your application when a user bookmarks the application to the home panel, you can do so by adding the JavaScript shown in Example 8-5 in the HTML page. We have an icon file located in the icons folder with the name `Finder.png`, which will be used as an icon for the application bookmark.

Example 8-5 JavaScript to add a custom icon

```
<link rel="apple-touch-icon" href="icons/Finder.png"/>
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-touch-fullscreen" content="YES" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black" />
```

8.6.5 Configuring the Ajax Proxy

We also needed to add an Ajax Proxy configuration to our WebSphere Application Server because we were supposed to make a connection to a remote ITM server located at a different IP address.

The role of the Ajax Proxy is to get a request from the sample.html page (client) and forward it to the remote ITM server and then forward the response received back from the ITM server to the client.

The Ajax Proxy configuration editor can be opened by double-clicking the **proxy-config.xml** file available in the **ITM Client** → **Web Content** → **WEB-INF** folder.

We added the context path “cms” for the ITM SOAP requests in the Ajax Proxy mapping and let it point to the ITM server’s IP address and port number we wanted to connect to. See Figure 8-8 on page 161.

Figure 8-8 Configuration of an Ajax Proxy

When using the above values, you can simply use the following address format to forward any request from the browser to the ITM server:

```
http://<<WAS server IP>>:<<WAS port number>> /ITMClient/proxy///cms/soap
```

This will result in a request to be automatically forwarded to the ITM server’s SOAP interface from the Ajax Proxy configured in WebSphere Application Server where our application is running.

8.6.6 Testing the application

The fastest way to test the application is to deploy it to the WebSphere Application Server test environment built into Rational Developer for System z. To do this, right-click the **ITMClient** application in the Project Explorer and select **Run As** → **Run on server ...** from the pop-up context menu.

You will then see a Run On Server pop-up window to select the WebSphere Application Server you want to deploy your application to. Select the server and click **Finish.**

The ITM client application will be published to the server and you can point to the address of the sample.html page from an Apple iPhone browser simulator program, or from a real Apple iPhone browser, as shown in Figure 8-9 on page 162.

Note: You can only test such an application if your phone has access to the server running the application. In the case of deployment to the WebSphere Application Server test environment in RDz on a Windows desktop, you would need wireless access to the network that your desktop is part of. In case of deployment to WebSphere Application Server on z/OS, you would need wireless access to the network that your z/OS server is part of.

Insert the ITM server's system administrator user ID and password and click **Managed System Details**. The user interface is shown in Figure 8-9 on page 162.

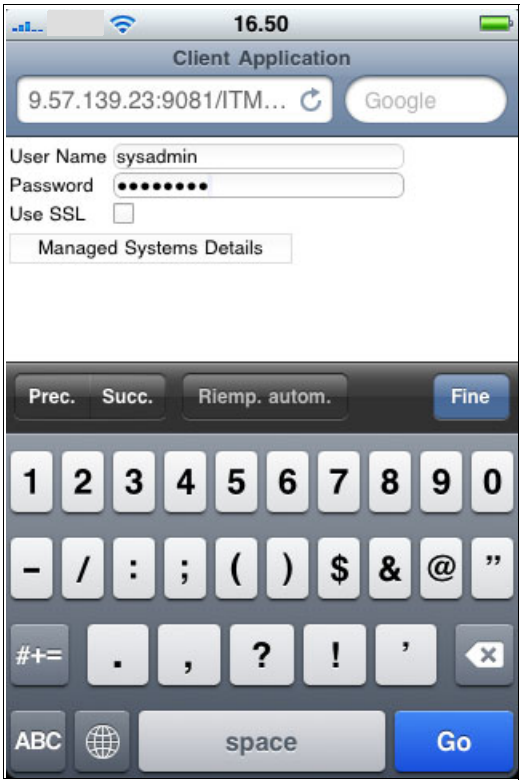


Figure 8-9 ITM client application's sample.html page in the Apple iPhone browser

Clicking this button will invoke a call to the JavaScript functions we explained earlier and the details of the systems will be requested from the ITM server. After processing the XML response received from the ITM SOAP server, the results are formatted and displayed in the Dojo grid already added to the page, as shown in Figure 8-10 on page 163.

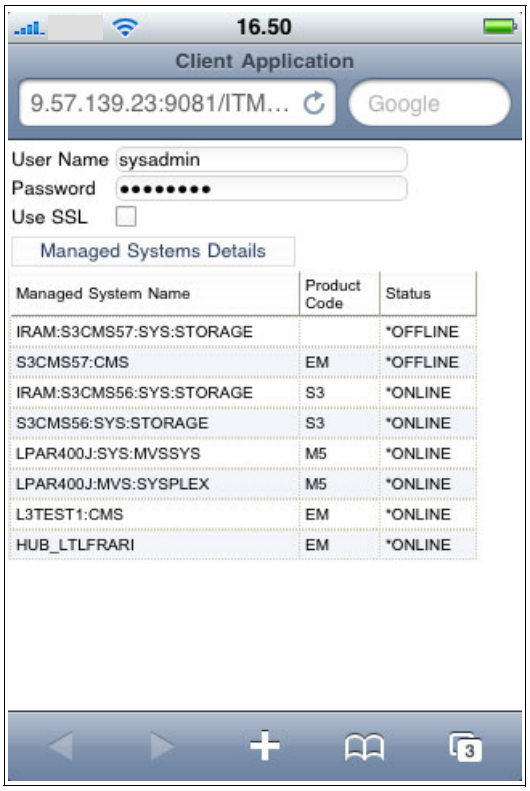


Figure 8-10 Information display about the ITM server's managed system

8.6.7 Deployment to WebSphere Application Server on z/OS

You can also deploy the JEE application to WebSphere Application Server on z/OS. Refer to Appendix C, "Deployment to WebSphere Application Server on z/OS" on page 197.

A

Test drive yourself

There are a few sample applications that you can execute yourself from your smartphone or PC browser and access a real z/OS back end. The only thing you need is a browser with Internet access over a Wi-fi network.

Start at:

`http://zserveros.demos.ibm.com:9080/zos/egl.html`

You will be presented a main menu, as shown in Figure A-1.

Figure A-1 Main menu

© Copyright IBM Corp. 2010. All rights reserved.

Important: We have tested these applications in a PC browser, on Apple iPhone with Safari browser and certain Nokia phones with Symbian. However, the panel optimization depends on your specific smartphone and its browser.

There are samples that access CICS and IMS transactions. There is also a sample invoking an RPG program on System i®.

The following scenarios are available in the “z/OS menu” at the time of writing:

- Mortgage CICS
- CICS/VSAM
- CICS/VSAM w/Google
- CICS/DB2 w/Google
- IMS sample
- CICS Account Sample appl.
- RPG/System i

Above scenarios are explained in a bit more in detail in “Description of the samples in the z/OS menu” on page 168.

There is also an example available now showing a simplified balance inquiry function with a link to Google maps. This sample is explained a bit more in “Bank example” on page 169.

Adding the z/OS menu to your home panel on your Apple iPhone

When Safari returns the data, use the **+** icon, located at the base of the Apple iPhone panel, to add this link to the Home panel, as shown in Figure A-2 on page 167. Then click **Add** shown in Figure A-3 on page 167.

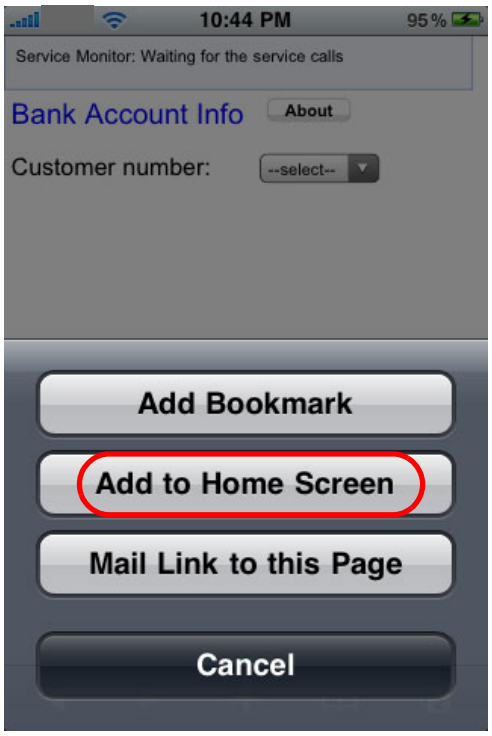


Figure A-2 Adding the z/OS main menu to the Home panel on Apple iPhone (1)

Figure A-3 Adding the z/OS main menu to the Home panel on Apple iPhone (2)

The icon will now be added to your Apple iPhone Home panel, which makes it much easier to use afterwards. See Figure A-4 on page 168.

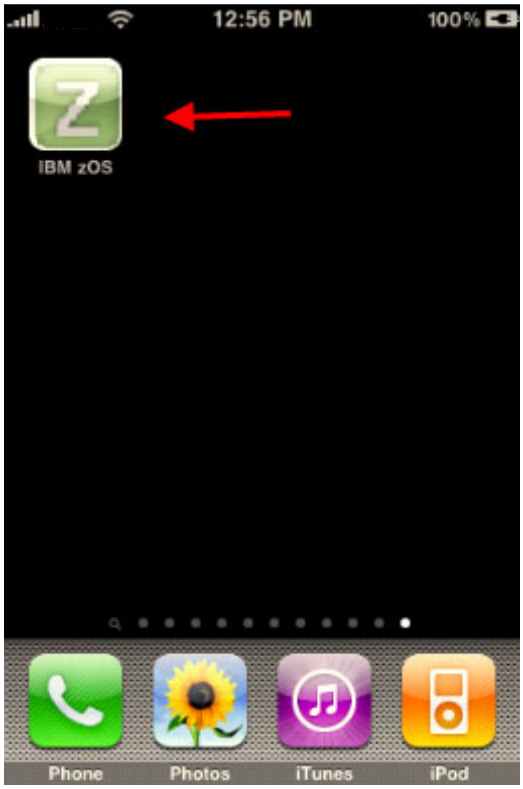


Figure A-4 z/OS application menu added to the Apple iPhone Home panel

Description of the samples in the z/OS menu

The following is a brief description of the samples. All samples are straightforward to operate and they all access a live z/OS back-end system at IBM, over the Internet.

Mortgage CICS

This is a COBOL/CICS application that calculates the monthly mortgage payment based on loan amount, interest % and duration. The graphic displayed is an example of using JavaScript. We used CICS Web Services for accessing the CICS application program.

CICS/VSAM

In this example we also use CICS Web Services to access the CICS back-end application. The CICS application, written in COBOL, accesses a VSAM data set on z/OS. The address read from the VSAM data set is passed to a Google map service. For details on the CICS implementation, see the paper at:

http://www.ibm.com/developerworks/websphere/techjournal/0909_barosa/0909_barosa.html

CICS/VSAM with Google

This is the same COBOL/CICS code as was used in the previous example, but the Google map service is invoked as soon as the data is retrieved, instead of having to press the button.

CICS/DB2

In this example the CICS Web service invokes a COBOL/CICS program that retrieves data from DB2. When selecting a customer using the drop-down, the address returned is passed to Google Maps services to show the customer location.

IMS sample

This is an example of accessing IMS data. The Web service in this example is running in WebSphere Application Server using Java connectors. Those connectors invoke a COBOL program running under IMS and the data is also retrieved from the IMS database.

CICS Account sample app

In this example you access CICS directly using the HTTP protocol. CICS provides all the client support through its HTTP server functionality. There is no middle tier in this example. This is another example to show how CICS can generate HTML and JavaScript to run in mobile devices.

RPG/System i

This is an application that runs on System i where the RPG code was wrapped into Web Services using Java connectors. This Java Web service is deployed in WebSphere Application Server running on System i. A small Java application is also deployed to z/OS to invoke System i. This is an example where two different operating systems are used. Click the question mark icon to see a picture with the architecture.

Click the **?** icon for details how the code was created and deployed, as seen in Figure 8-11.

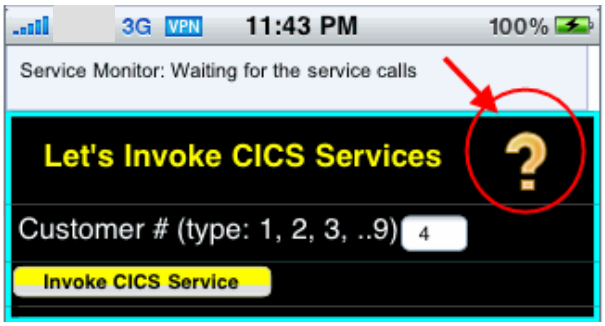


Figure 8-11 Question mark icon

Also click the green arrow in the top left corner to check the response time.

Bank example

Another sample that can be used directly is the Bank sample. You can invoke this sample by entering the following on your phone:

`http://zserveros.demos.ibm.com:9080/bank/egl.html`

Again, you can add this sample to your Home panel as explained earlier and as shown in Figure A-5 on page 170 and Figure A-6 on page 170.

Note: Just remember to select **Add to Home Screen**. Adding to Bookmark will not create the icon in the iPhone homepage.

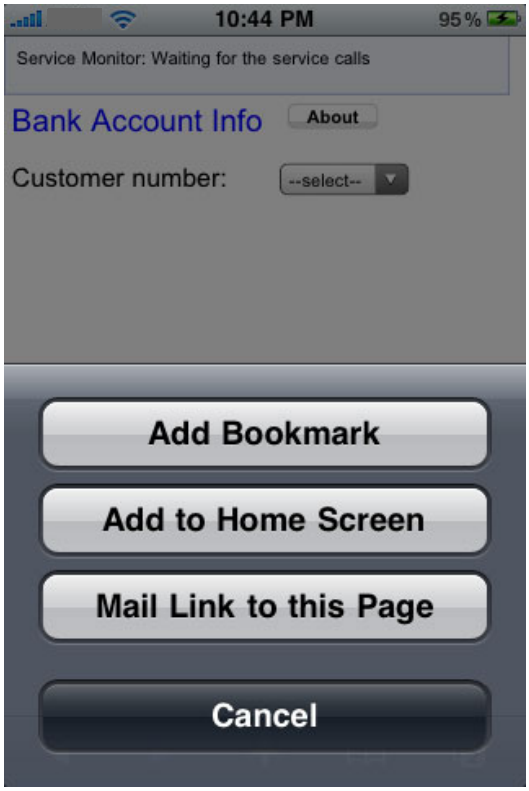


Figure A-5 Adding the Bank sample to Apple iPhone Home panel (1)

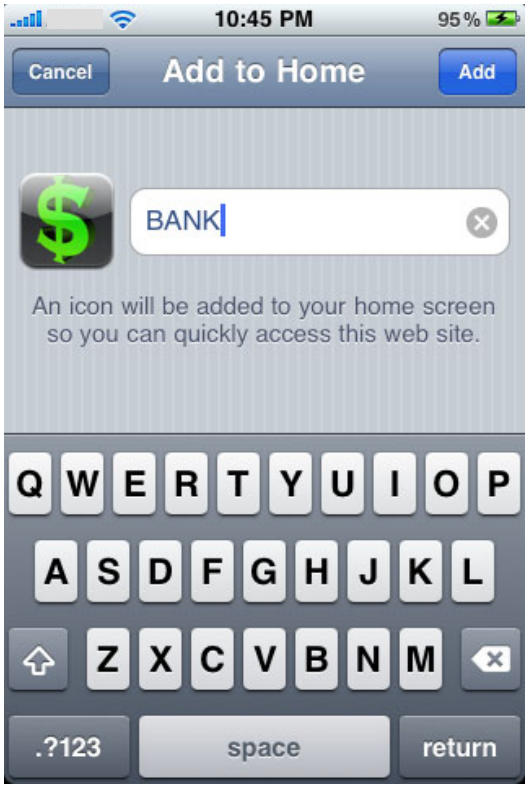


Figure A-6 Adding the Bank sample to Apple iPhone Home panel (2)

To use this sample, select a customer number from the drop-down list, as shown in Figure A-7.

Figure A-7 Selecting a customer number

Clicking **About** will show you some explanation about the smartphone client that had been developed. See Figure A-8 on page 172.

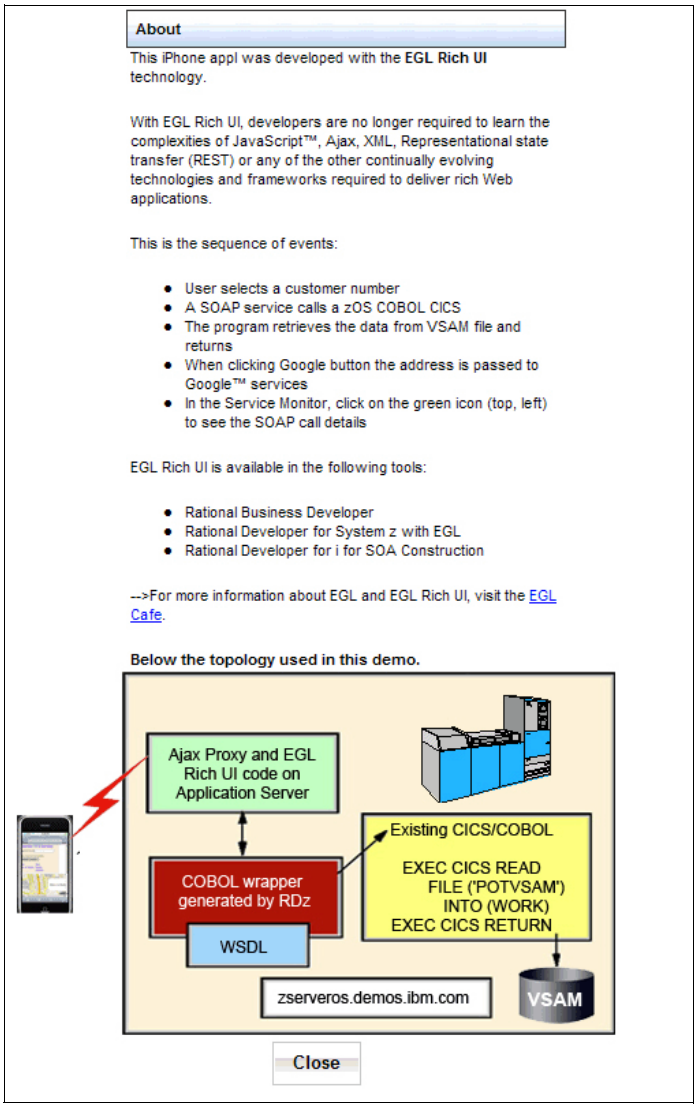


Figure A-8 About the window for the Bank sample

After selecting a customer number, you will get a panel showing the account balance. There is also a button you can click to show a Google map with the address of the person. See Figure A-9 on page 173.

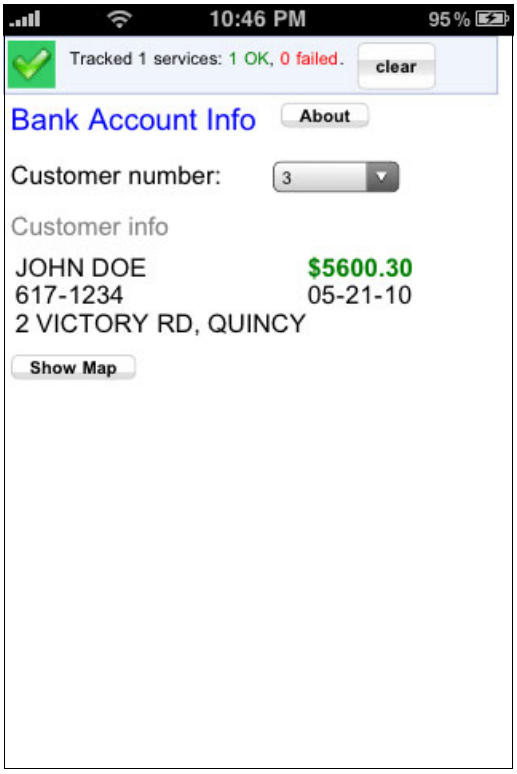


Figure A-9 Account balance returned

After clicking **Show Map** a Google map service is invoked, displaying a map with the address of the account holder. See Figure A-10.

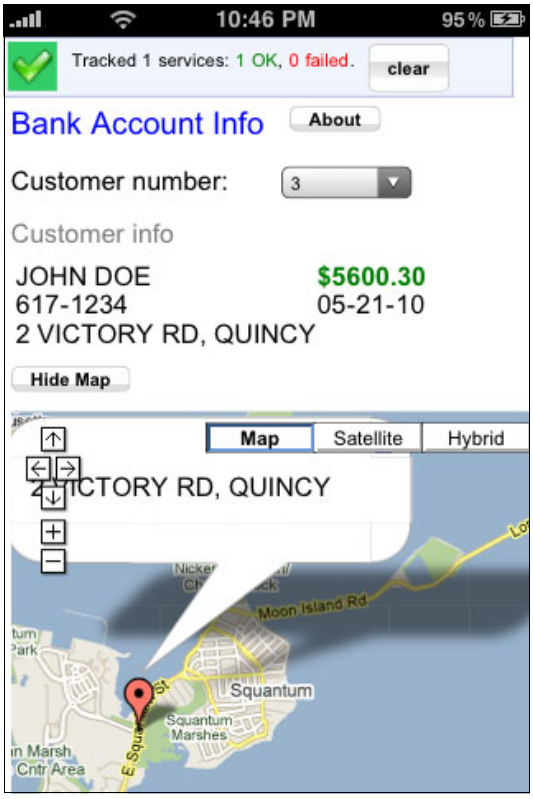


Figure A-10 Account holder with Google map

Again, the response time of this interaction is very good. When clicking the button, as shown in Figure A-11, you will see a window with the connection details, as shown in Figure A-12. The response time is shown in milliseconds in blue.

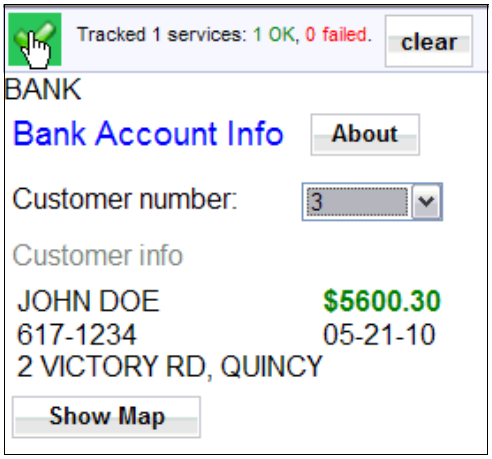


Figure A-11 Clicking the button to get the connection details

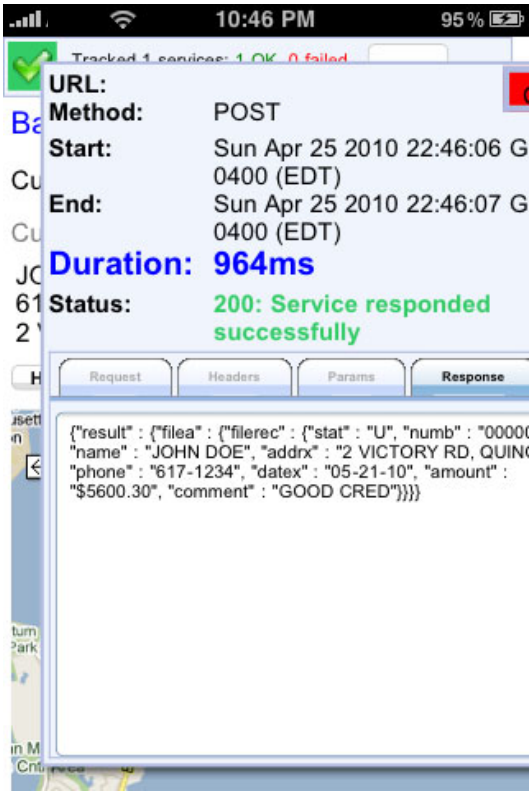


Figure A-12 Response time

A closing word on the benefits of using EGL for Rich UI

The smartphone client application in the samples shown in this appendix have been developed with EGL Rich. EGL Rich UI has many benefits over lower-level implementations (like JavaScript or AJAX). Among them, EGL:

- Reduces development costs, accelerates time to market, and helps deliver new, higher quality Web 2.0-based business solutions, due to the higher productivity and simplicity involved.
- Has very good performance with a rich user interface.
- Requires a shorter learning curve and lesser training costs; developers only need to learn one language (EGL) in order to build back-end business logic and services, and front-end user interfaces.
- Supports multiple browsers (write once, run in most popular browsers).
- Extends the value and the life of the reliable business logic that resides in System z and i systems by enabling applications that run transactional services within innovative applications that improve user experience and productivity.
- Seamlessly integrates enterprise information and processes with other sources of information, data, and services across the enterprise or on the Web to better serve users, customers, and partners (through mashups).
- Leverages emerging SOA-based industry-vertical services provided by vendor and free or fee powerful services (for example, Google Charts, Google Maps, Xignite, and so on).
- Lowers the advanced skills barrier and leverages traditional business-oriented developers while delivering innovative and powerful state-of-the-art solutions to the business.

B

IBM Tivoli Monitoring and the SOAP interface

This appendix discusses the IBM Tivoli Monitoring architecture and its SOAP interface.

© Copyright IBM Corp. 2010. All rights reserved.

ITM architecture

In order to effectively communicate with the ITM SOAP server and obtain data from the monitoring agents that it hosts, you need to know a little ITM terminology and a little about how the various components of ITM fit together and relate to one another.

ITM consists of the following major components, shown in Figure B-1.

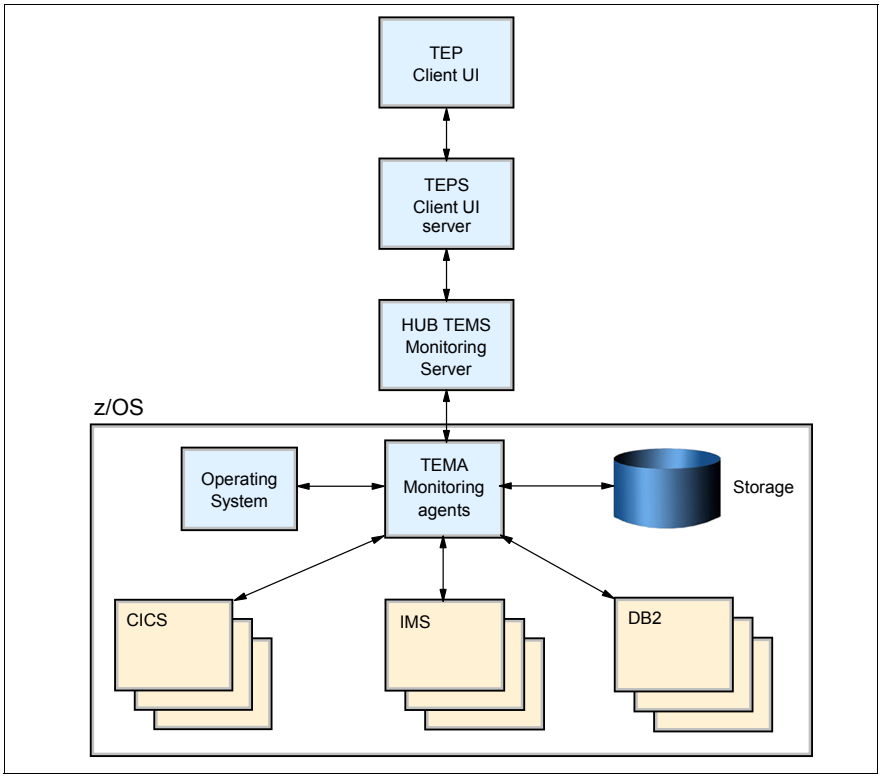


Figure B-1 ITM architecture

Tivoli Enterprise Portal (TEP)

The IBM Tivoli Monitoring client, Tivoli Enterprise Portal (TEP), is a Java-based user interface for viewing and monitoring your enterprise network. Depending on how it was installed, you can start Tivoli Enterprise Portal as a desktop application or through your browser as a Web application.

Tivoli Enterprise Portal Server (TEPS)

The Tivoli Enterprise Portal client connects to the Tivoli Enterprise Portal Server (TEPS). The Tivoli Enterprise Portal Server is a collection of software services for the client that enables retrieval, manipulation, and analysis of data from the monitoring agents on your enterprise.

Tivoli Enterprise Monitoring Server (TEMS)

The Tivoli Enterprise Portal Server connects to the main, or hub, Tivoli Enterprise Monitoring Server (TEMS). The monitoring server acts as a collection and control point for alerts received from the monitoring agents, and collects performance and availability data from

them. The hub monitoring server correlates the monitoring data collected by monitoring agents and any remote monitoring servers and passes it to the portal server for presentation in the portal console.

Tivoli Enterprise Monitoring Agents (TEMAs)

Tivoli Enterprise Monitoring Agents (TEMAs) are installed on the systems or subsystems whose applications and resources you want to monitor. An agent collects monitoring data from the managed system and passes it to the monitoring server to which it is connected. The client gathers the current values of the attributes and produces reports formatted into tables, charts, and relational table-based topology views. It can also test the values against a threshold and display an alert icon when that threshold is exceeded or a value is matched. These tests are called situations.

An ITM configuration is not limited to a single TEMA on a z/OS LPAR. For example, the monitoring agents might be split amongst multiple TEMAs, as shown in Figure B-2.

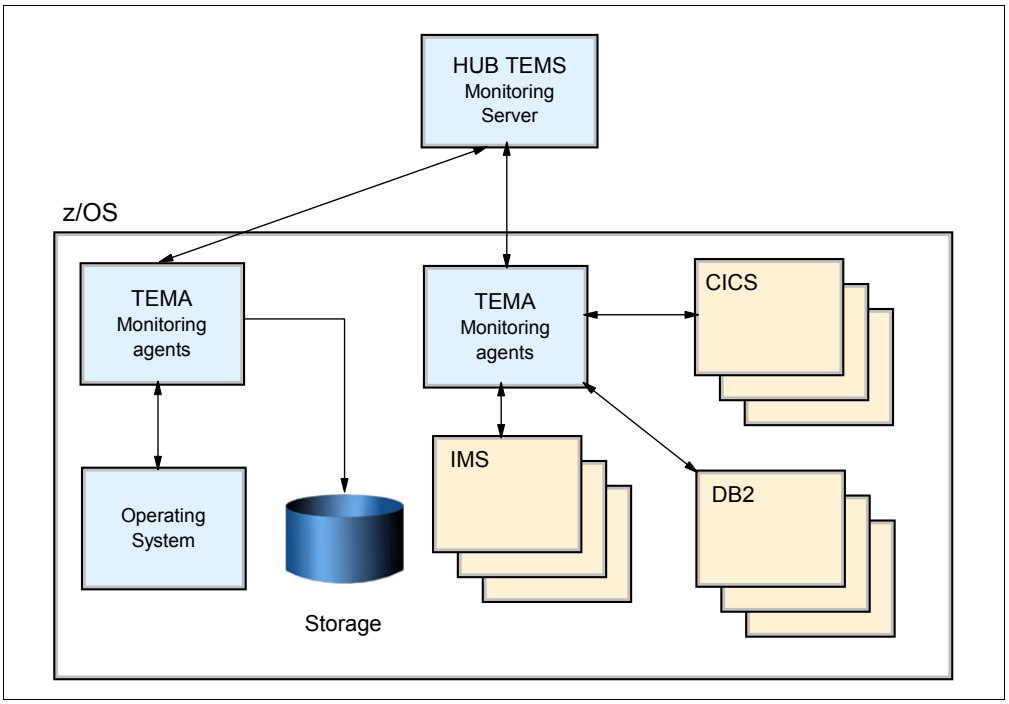


Figure B-2 TEMAs

In this configuration, multiple TEMAs are running on the z/OS LPAR. One is running agents that are monitoring the operating and storage subsystems while the other is running agents that are monitoring the CICS, IMS, and DB2 subsystems on this LPAR.

Remember that this arrangement of TEMAs, each running agents for the products being monitored, is repeated for each z/OS LPAR within the z/OS sysplex. However, all TEMAs ultimately communicate with a HUB TEMS. The HUB TEMS may be on a z/OS LPAR, a zLINUX partition or completely off platform, for instance on a Windows Server,

ITM SOAP server location

You can configure a SOAP server to run on the HUB TEMS and also on the TEMAs on the systems being monitored. However, if you connect to the SOAP server on a monitored system you can only see data that is being collected by agents connected to, or running within, that TEMA.

Therefore it makes far more sense from a data availability point of view to define your SOAP server as running in the hub TEMS. That way you can see data from all of the agents connected to it.

For the purpose of this discussion, we are assuming a configuration similar to that shown in Figure B-3 with the SOAP server interface being defined within the HUB TEMS. The IP address of the SOAP server will be the IP address of whatever machine the HUB TEMS is executing on. See Figure B-3.

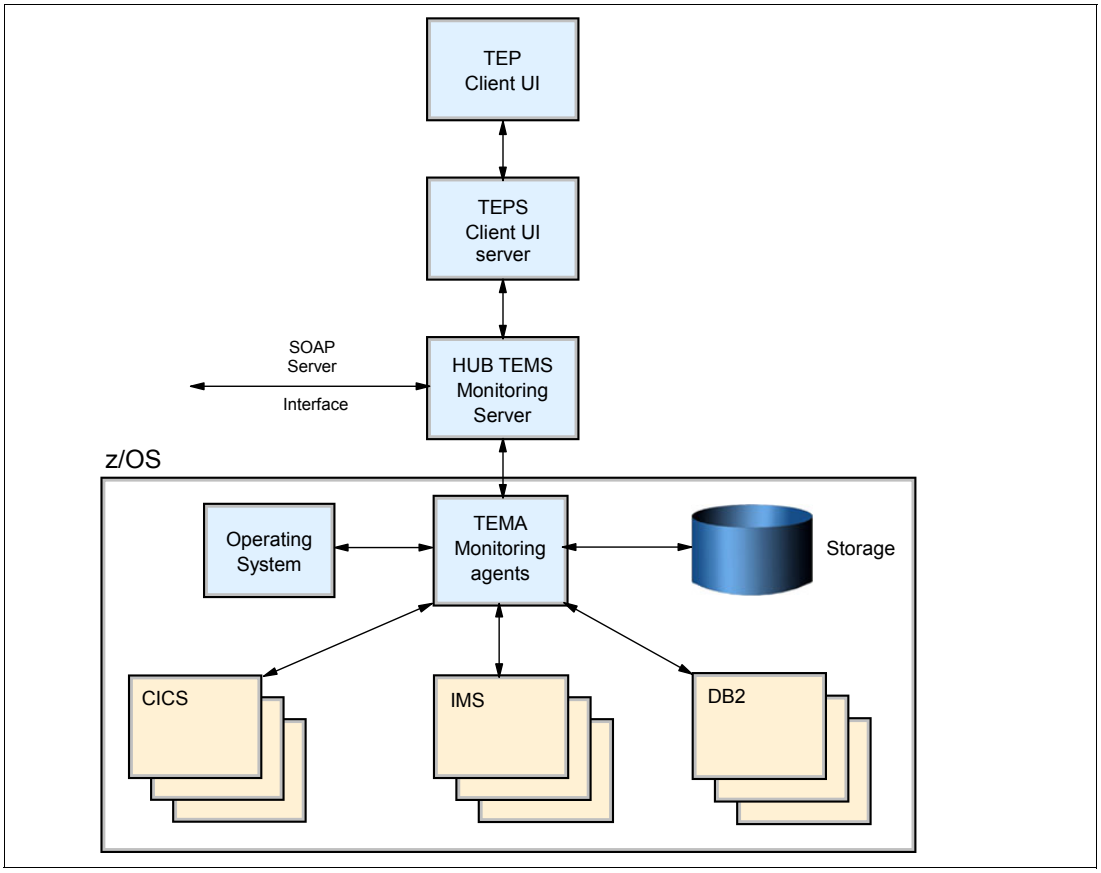


Figure B-3 ITM SOAP Server location

Managed systems and managing systems

What is a *managed system*? A managed system is any component that is being monitored by an ITM agent. Thus, if you have the z/OS operating system agent installed, the managed system is the operating system itself.

If you have the CICS product installed, then each CICS address space on the same z/OS LPAR that the agent is monitoring is a managed system (it might not be monitoring them all, depending upon how the product and CICS are configured).

A managed system name is the monitoring agent's interpretation and mapping of some aspect of the monitored component, for example the CICS job or started task name, into a format that is registered with and known to ITM.

When we make SOAP requests to ITM to obtain monitoring data we are really making requests to a specific agent, identified by the managed system name. Since that agent is monitoring a specific component such as a CICS region, it returns only the data for that component.

Also associated with a managed system is the *managing system*. This is the ITM component that is managing the manager. Each ITM managing component, all the way up to the hub, has a manager. Together these form a hierarchy of parent and child managers all the way from the hub down to the actual agent for the monitored application.

If we examine the managed systems of an ITM installation we can see that the managed system and managing system names form a hierarchy from the hub down to the agents. See Table B-1.

Table B-1 Managed system and managing system hierarchy

Managed System	Managing System	Description
HUB_LTLFRARI	HUB_LTLFRARI	HUB
L3TEST1:CMS	HUB_LTLFRARI	TEMA
LPAR400J:SYS:MVSSYS	L3TEST1:CMS	z/OS LPAR Agent
IRAM:S3CMS56:SYS:STORAGE	L3TEST1:CMS	IRA Manager for Storage
S3CMS56:SYS:STORAGE	IRAM:S3CMS56:SYS:STORAGE	z/OS Storage Agent
LPAR400J:MVS:SYSPLLEX	L3TEST1:CMS	z/OS Sysplex Agent

Pictorially this can be as shown in Figure B-4.

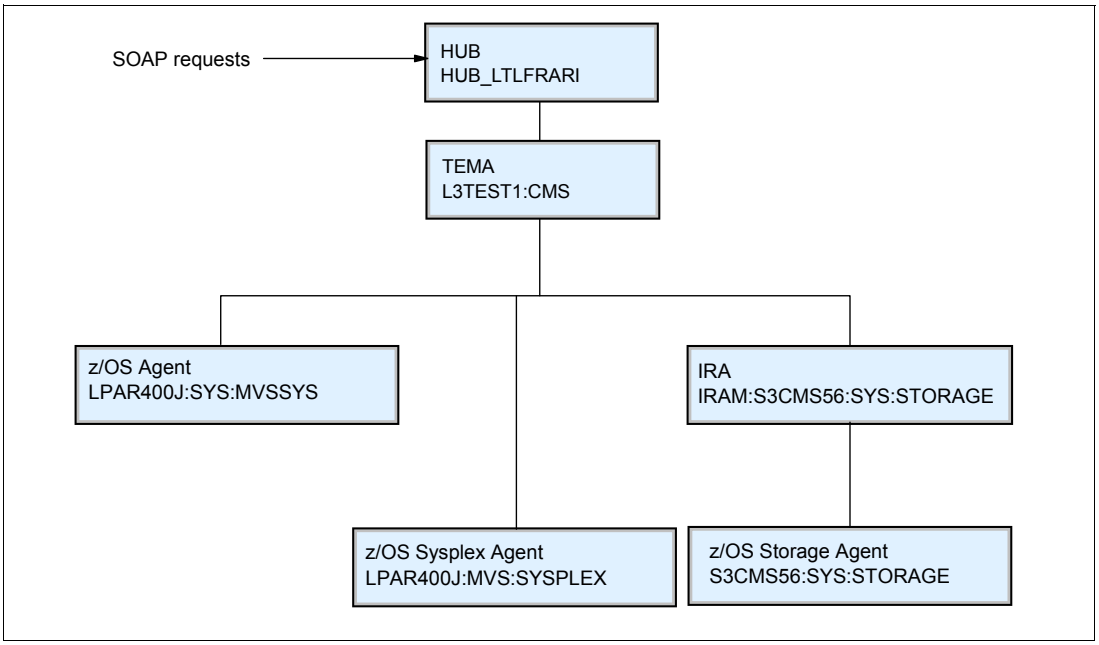


Figure B-4 Managed system and managing system hierarchy

You can see that a SOAP request arrives at the HUB TEMS and must be sent, via the ITM architecture, to the appropriate agent. You identify the agent that you want to request data from, within the body of the SOAP request itself.

Product codes and managed system names

It is necessary to understand a little bit about the ITM product codes if you want to determine which managed systems you want to issue requests too, especially if you want to do this programmatically.

So, what are *product codes*? Each ITM monitoring component, for example Omegamon XE on z/OS (the operating system monitor) or Omegamon XE for Storage (the z/OS Storage system monitor), is assigned a two-character product code. In addition, the ITM components themselves also have product codes. You can see these product codes in the list of available managed systems in the ITM TEP Managed System Status workspace, part of which is shown in this panel shot of the ITM Managed System status view (Figure B-5).

Managed System Status

Status	Name	Product	Version	Managing System	Timestamp
*ONLINE	LPAR400J:MVS:SYSPLEX	M5	04.20.00	L3TEST1:CMS	02/08/10 17:41:01
*ONLINE	IRAM:S3CMS56:SYS:STORAGE	S3	04.20.02	L3TEST1:CMS	02/08/10 17:41:30
*ONLINE	S3CMS56:SYS:STORAGE	S3	04.20.02	IRAM:S3CMS56:SYS:STORAGE	02/08/10 17:41:30
*ONLINE	LPAR400J:SYS:MVSSYS	M5	04.20.00	L3TEST1:CMS	02/08/10 17:41:52
*ONLINE	L3TEST1:CMS	EM	06.21.00	HUB_LTLFRARI	02/08/10 19:47:00
*ONLINE	HUB_LTLFRARI	EM	06.21.00	HUB_LTLFRARI	02/08/10 19:47:59

Figure B-5 Product codes as shown in the ITM TEP Managed System Status workspace

You can see in Figure B-5 that there are three product codes, EM, S3, and M5. If you look at a typical navigation tree in the TEP (Figure B-6 on page 183) you will see the managed system names and a description of what they are. For example, the S3CMS56:SYS:STORAGE managed system is under the Storage Subsystem node on the

tree and therefore is part of the Omegamon XE for storage product. Referring back to the Managed System status view (Figure B-5), you can see that the product code for that managed system name is S3. Thus you can determine that managed systems with the S3 product code are part of the Omegamon XE for Storage product. In the same way you can determine that the M5 product code belongs to the Omegamon XE on z/OS product.

But what about the EM product code and why are there two managed systems for the M5 product and two for the S3 product, but only one managed system name for the storage and z/OS products on the navigation tree? (Figure B-6 on page 183)

Well, the EM product is part of the ITM infrastructure. You could probably figure this out just from the fact that the entries have both the managed system and managing system names as HUB_LTLFRARI, which is the HUB TEMS.

But what about the storage product? It has two managed system names but only one entry on the navigation tree. If you look at the diagram of the managed system hierarchy you can see that the S3CMS56:SYS:STORAGE managed system name is beneath the IRAM: S3CMS56:SYS:STORAGE component. The IRAM:S3CMS56:SYS:STORAGE component is part of the infrastructure and not a managed system. Thus, as far as the SOAP request goes, we can ignore it.

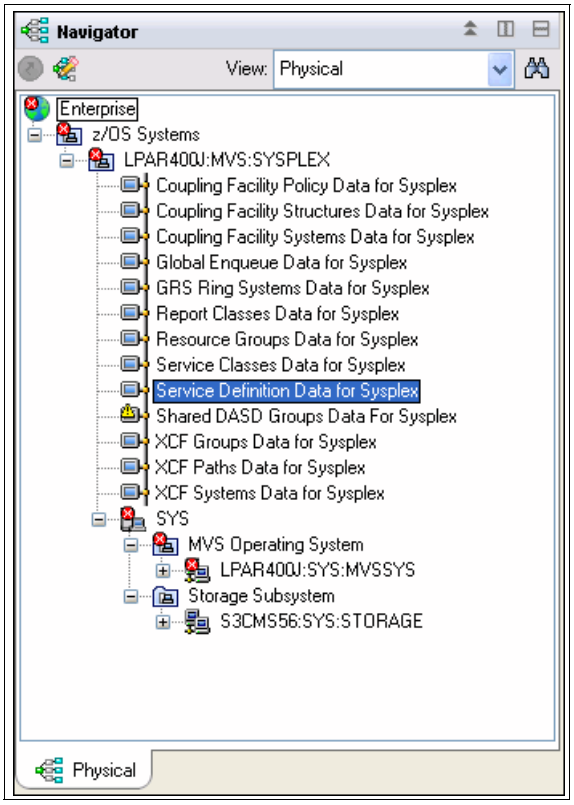


Figure B-6 Navigation tree

Now, what about the M5 Omegamon XE on z/OS product? It has two managed system names, LPAR500J:SYS.MVSSYS and LPAR400J:MVS:SYSPLEX.

If you look at the navigation tree (Figure B-6) you can see the LPAR500J:SYS.MVSSYS entry under the MVS Operating System node. Thus we can determine that this is the LPAR level agent. So what is the other one? Again, looking at the navigation tree you can see LPAR400J:MVS:SYSPLEX under the z/OS systems node. This entry represents the Sysplex. So if you wanted to send a SOAP request to the LPAR level agent you would have to direct it

to the LPAR500J:SYS.MVSSYS managed system and if you wanted to obtain data from the sysplex agent you would have to direct it to the LPAR400J:MVS:SYSPLEX managed system.

Managed System Status

Looking at the Managed System Status view tree again (Figure B-5 on page 182) you can see that there is a Status column. This indicates the managed systems status, ***ONLINE** or ***OFFLINE**. Obviously, in order to be able to request data from a managed system agent, the managed system itself must be online.

Why all this is important

Well, if nothing else you need to be able to determine the status of a managed system before you try to send a request to it since you obviously cannot request data from an offline managed system. But just as important is that if you want to be able to programmatically determine which managed systems belong to which products and in the situations where there are apparently different formats of managed system names, which ones represent real managed systems and which ones represent infrastructure components, then you need to be able to recognize which are which from the formats of the names or from the hierarchy of managed systems.

You can see that the real managed systems are at the end of the hierarchy of managed systems and that the hub points to itself. In addition, you can see that from a management agent point of view both the z/OS Sysplex agent (LPAR400J:MVS:SYSPLEX) and the z/OS LPAR Agent (LPAR400J:SYS:MVSSYS) are at the same logical level as children of the L3TEST1:CMS component and are both end points on the hierarchical chain of managed systems. This is true even though we know that LPARs are logically part of the Sysplex. From an ITM managed system point of view, both are managed systems and are logically at the end of the chain. They appear in the correct order on the TEP because the interface knows the relationship between the components and can arrange them in the correct order on the navigation tree.

Our first ITM SOAP request

If you point Internet Explorer (this only works in IE) to:

```
http://itm_host:1920///cms/soap/kshhsoap.htm
```

where **itm_host** is the IP address or domain name of the computer hosting the HUB TEMS, then you should see the Generic SOAP Client as shown in Figure B-7 on page 185.

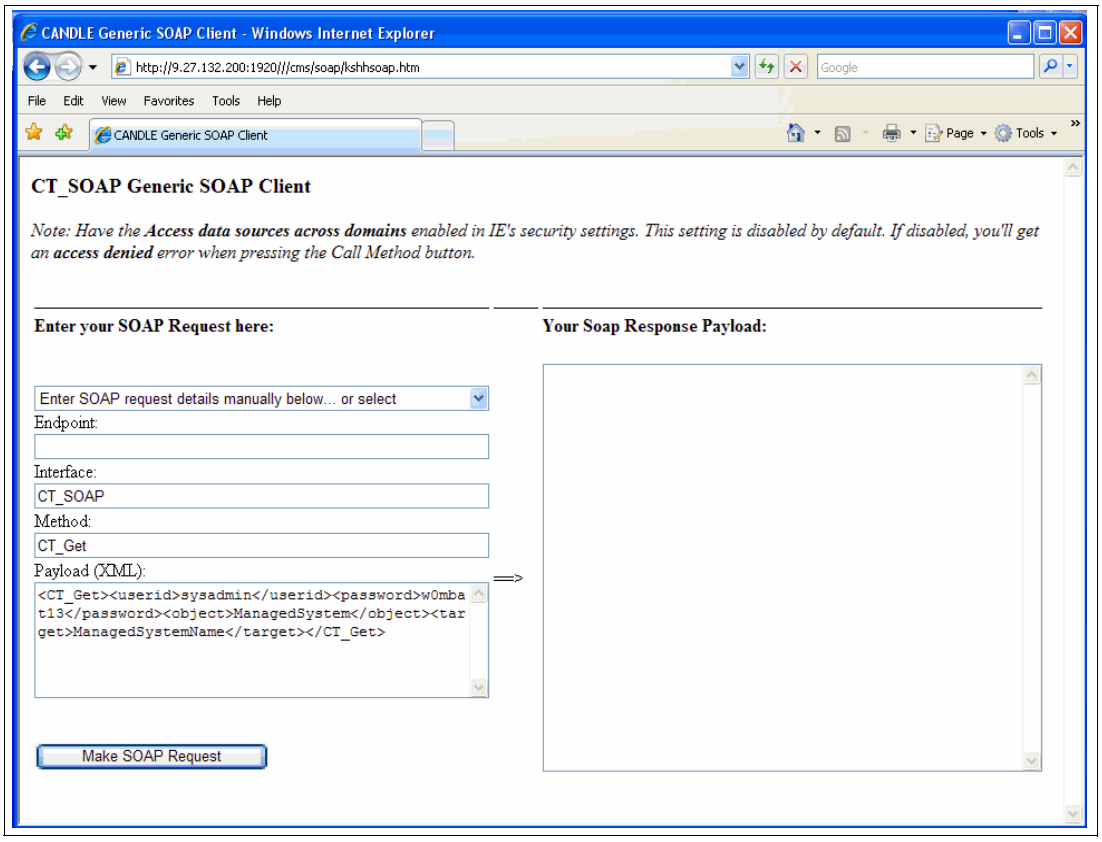


Figure B-7 ITM Generic SOAP Client

This Web page enables you to make SOAP requests to the ITM SOAP server. You will need a user ID and password (if ITM is configured to require a password; it may not be) for the ITM system that you are making the request to.

Click the **Enter SOAP details manually below… or select** drop-down box and select **Get Object CT Method**.

The Endpoint will change to something like:

```
http://localhost:1920///cms/soap
```

Unless you are actually on the same physical computer as the ITM HUB TEMS and SOAP server is located on, you will need to change “localhost” to the IP address or hostname of the computer hosting the HUB TEMS. the same as you entered for itm_host in the address bar.

In the Payload (XML) field is an initial SOAP request. This particular request will obtain the list of managed systems from the SOAP server. You may need to change the user ID between the `<userid>` and `</userid>` tags if your userid is not sysadmin, and you will need to enter your password for the userid between the `<password>` and `</password>` tags if the ITM system requires a password. This is a typical default request, as shown in Example B-1 on page 186.

Example B-1 Initial SOAP request

```
<CT_Get>
<userid>sysadmin</userid>
<password>mypasswd</password>
<object>ManagedSystem</object>
<target>ManagedSystemName</target>
</CT_Get>
```

Click **Make SOAP Request** to send the request to the ITM SOAP server. If all goes well, you will see the XML output in the text area below the Your SOAP Response Payload heading.

`<object>ManagedSystem</object>` is what we are requesting, which is the list of managed systems. This is also known as the “attribute group” name.

`<target>ManagedSystemName</target>` is where the request is going to—basically, the HUB in this case.

The SOAP response header

Example B-2 shows a typical SOAP response header down to the point where the data starts.

Example B-2 SOAP response header

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
     <SOAP-CHK:Success
      xmlns:SOAP-CHK = "http://soaptest1/soaptest/"
      xmlns="urn:candle-soap:attributes">
         <TABLE name="O4SRV.INODESTS">
         <OBJECT>ManagedSystem</OBJECT>
```

The following is of interest in the header:

The `SOAP-CHK:Success` tag indicates that the request was successful, at least as far as the SOAP server was concerned. What this really means is that the user ID (and password if included) is valid, the request is valid XML and that it recognized the request `CT_Get` in the root tag.

The object name that we requested is returned within the `<OBJECT>` tag, and the internal table name that corresponds to the object name is returned in the name attribute of the `TABLE` tag.

Should the request fail, the response body looks as shown in Example B-3 on page 187.

Example B-3 SOAP response if failed

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
          <faultcode>SOAP-ENV:Client</faultcode>
          <faultstring>CMS logon validation failed.</faultstring>
          <detail>sysadmin</detail>
        </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Notice that `SOAP-CHK:Success` has been replaced by `SOAP-ENV:Fault`. The `<faultstring>` tag contains a short (often cryptic) description of why the request failed and there may be additional data in the `<detail>` tag. I forced this request to fail by specifying an incorrect password for the userid. And that is pretty much the extent of the error information returned by the SOAP server.

The actual DATA

Assuming the request got past the initial checking by the SOAP server, it gets passed to the agent. The data is returned (as a child node of the object tag in the response XML) with the layout as shown in Example B-4.

Example B-4 Data returned

```
<DATA>
   <ROW>
      <attrname>value</attrname>
   <attrname>value</attrname>
         :
         :
   </ROW>
   <ROW>
      <attrname>value</attrname>
   <attrname>value</attrname>
         :
         :
   </ROW>
</DATA>
```

If the agent returns NO data (unlikely with the request for the list of managed systems, but possible with agents for real managed systems) then there is a NO DATA element in the response body. However, the response in the SOAP header is still `SOAP-CHK:Success`.

Assuming you get data back, it is arranged within repeating ROW elements, with the actual data being contained within attributename tags. “Attributename” is not actually attributename but is replaced by the name of each attribute within the “attribute group”.

So, for example, a ROW of data from the ManagedSystem attribute group might look as shown in Example B-5.

Example B-5 ROW of data

```
<ROW>
<Timestamp>1100208174101002</Timestamp>
<Name>LPAR400J:MVS:SYSPLEX</Name>
<Managing_System>L3TEST1:CMS</Managing_System>
<ORIGINNODE>LPAR400J:MVS:SYSPLEX</ORIGINNODE>
<Reason></Reason>
<Status>*ONLINE</Status>
<Product>M5</Product>
<Version>04.20.00</Version>
<Type>V</Type>
<HLOCFLAG></HLOCFLAG>
<Host_Info></Host_Info>
<HHOSTLOC></HHOSTLOC>
<Host_Address></Host_Address>
</ROW>
```

This ROW represents ONE managed system, in this case the name (in the ”name” node) is LPAR400J:MVS:SYSPLEX. You can see the managed system status and product code in the data along with other information about the managed system.

In the output for this SOAP request, there will be one ROW of data per managed system that is registered with this ITM hub. Notice also how the child node names within the ROW element indicate what the data within the node is, Status, Product, Name, and so on. There is no “tell me what attributes are in an attribute group” SOAP request and there is no form of dictionary returned as part of the response.

Basically you have to examine the node names of the attributes returned within a ROW in order to determine what the data is. On the plus side, every ROW returned will be identical and will contain the same set of attribute node names. However, if the agent returns no data, you have no way to determine, from the response at least, what attributes ”would” be returned had the agent returned any data. In other words, you need to actually receive data back from a request in order to determine what the request will return.

Decoding the managed system list

Earlier we talked about the hierarchy of managed systems within the ITM environment and how you need to know the name of a managed system in order to send a SOAP request to it.

Now we can obtain a list of the managed systems from the ITM SOAP server, but how can we use that to determine the names of the actual managed systems we can issue a request to? Remember that not all managed systems are ”real” managed systems; some are part of the infrastructure.

There are three techniques that we can think of. These are:

- Go through the list and build a tree structure of the managed systems using the name in the ManagingSystem nodes to determine which node owns which node. The end points are the managed systems with no children and these represent the real managed systems. For example, the S3 product has two managed system names but only one, S3CMS56:SYS:STORAGE is an end point and therefore only it is a true managed system.

  The M5 product has *two* managed systems that do not have child nodes and therefore both of these are managed systems that you can request data from. Everything else in our

example environment is part of the infrastructure. However, without additional processing you cannot determine which is the sysplex entry and which is the LPAR entry.

- Use the product code in the Product tag and a pattern matching mechanism to determine which managed system is a real one and, as in the case of the Omegamon XE on z/OS (M5) product, which managed system represents the sysplex and which, the LPAR.

  For the M5 product the name of the sysplex-managed system always ends with the string :MVS:SYSPLEX. The first node, LPAR400J in this case, of the name is the actual sysplex name, although the user can override that in the TEMA configuration step.

  Managed systems with a name of the form “name1:name2:MVSSYS” are LPAR-level managed systems. name1 is the sysplex name and enables you to match the managed system to its owning sysplex-managed system and name2 is the SMF ID of the system that the agent is running on. Again the user can change this during TEMA customization.

- A hybrid approach combining both of the above techniques. Map the nodes using the technique described in the first bullet and then apply pattern matching to the end nodes (managed systems) to determine what they are, for example a sysplex entry or an LPAR entry for the M5 product.

In order to send the correct request to the correct managed system type for products such as the Omegamon XE on z/OS (M5) product, it is necessary to do some sort of pattern matching to detect which is which. However, if you wanted to display for instance a logical layout of the managed systems using the information from the managed systems SOAP query, then you would still need to map the relationships between the managed systems as well.

The advantage of applying some logic to the decoding process is that you can also add some value to the data. For example, we know that any managed system for the M5 product whose managed system name is of the form “name:MVS:SYSPLEX” is the sysplex-managed system entry for sysplex name. Therefore, when we decode this information we can add data that describes this more completely to the user. Instead of just seeing name:MVS:SYSPLEX, we can add a description to indicate that this is a sysplex entry. We can also use our knowledge of the fact that LPAR entries share the same sysplex name in the first qualifier to group the LPAR entries under the owning sysplex entry in the output display. This makes it easier for the user to navigate around the list of managed systems on the device. Remember, we are intending to display this information on a smart phone type device so usability has to be one of our primary concerns. It is no use overloading the user with a flood of unordered data.

Attributes group

We have already talked about attribute groups a little bit. An *attribute* is a metric or component name, for example a job name, job number, CPU usage, or storage used. *Attribute groups* are collections of attributes that are returned by an agent. For example, information about an address space on a z/OS system. We have already seen that the SOAP server returns each attribute group as a ROW element within the response XML with the attributes as child nodes of the ROW element. If the monitoring agent returns multiple rows of data, for example, information about all the address spaces on an LPAR, then the SOAP response data will contain multiple ROW elements, each containing the same set of attribute names.

We have also seen that the SOAP server returns the attribute name as the actual node name of the elements within each ROW element. So, for example, a ROW element with a SOAP response might look as shown in Example B-6 on page 190.

Example B-6 ROW element

```
<ROW>
   <Job_Name>CICSPROD</Job_Name>
   <Job_Number>STC00123</Job_Number>
   <CPU_Use>12345</CPU_Use>
   etc;
</ROW>
```

`Job_Name`, `Job_Number` and `CPU_Use` are just some of the attributes of this group and the complete ROW element is just one instance of the result set returned by the agent. The complete set of attributes for a ROW is an “attribute group”.

So, how do you request a specific attribute group?

When we requested the list of managed systems, the request body looked as shown in Example B-7.

Example B-7 List of managed systems request body

```
CT_Get>
<userid>sysadmin</userid>
<password>mypasswd</password>
<object>ManagedSystem</object>
<target>ManagedSystemName</target>
</CT_Get>
```

Within this request, the content of the `<object>` tag is the name of the attribute group that is being requested. So all we need to do is replace ManagedSystem with the name of the attribute group we want and off we go. Well, not quite, but that is the first step.

The question is, how do you know what attribute group name to place within the “object” tag? This is where working with the ITM SOAP interface gets tricky. From a practical standpoint wading through large amounts of documentation, even online is difficult and documentation quickly becomes out of step with live systems. That is just a fact of life when it comes to software. The most accurate method you can use to determine the available attribute groups is programmatically. However, there is no published API for this.

This leaves us with two methods that we have found that work. The first, which we will not describe in detail, requires that you use the TEP user interface to create a new query for a product section. Whilst doing this you have to select the attributes you want to use in the query and the selection dialog contains the attribute group name, albeit formatted for better viewing. Basically, at least in the English Language version of the TEP that we have used, underscores in the attribute group name are replaced by spaces. Thus, a name that is displayed as for example Address Space CPU Utilization would actually be Address_Space_CPU_Utilization.

The second, and now our preferred method, is to use a freely available IBM toolkit, known as *ITMsuper*. This is a really nice Internet Explorer-only based tool developed by Arun Desai of IBM and available on the IBM Tivoli Open Process Automation Library (Opal) at:

`http://www-01.ibm.com/software/brandcatalog/portal/opal`

Just search for ITMSuper. In order to download it you will need to sign up for a universal IBM user ID, but that is free. Once you have downloaded the application, unzip it and run the `itmsuper.hta` file in the `itmsuper` directory. This is a windows HTML Application file, basically

a desktop Web application. You will be prompted to sign in and specify the domain name or IP address of the ITM SOAP server that you want to use.

ITMSuper will then attempt to obtain a list of all the managed systems by product code, as shown in Figure B-8.

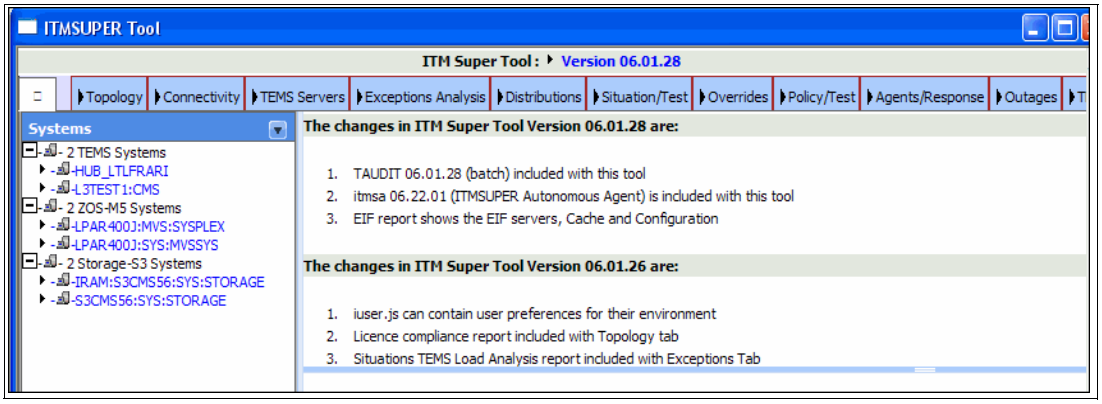


Figure B-8 ITMSuper with a list of managed systems

Clicking one of the managed systems will return a list of the attribute groups defined for that managed system, as shown in Figure B-9.

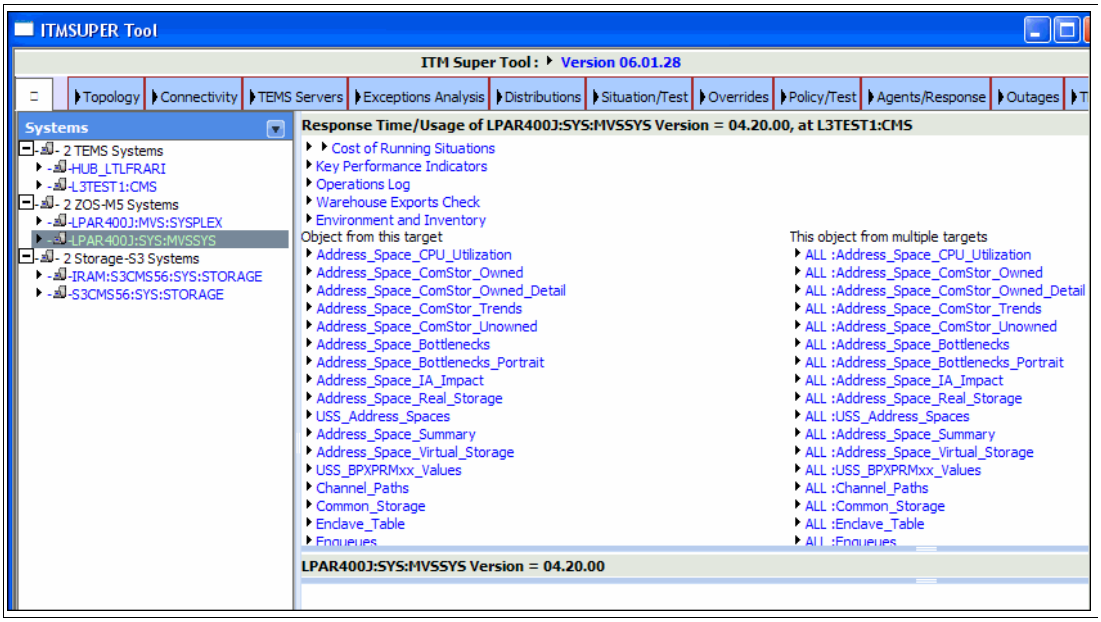


Figure B-9 ITMSuper with attribute groups

Clicking an attribute group requests the data for that attribute group for the managed system, as shown in Figure B-10 on page 192.

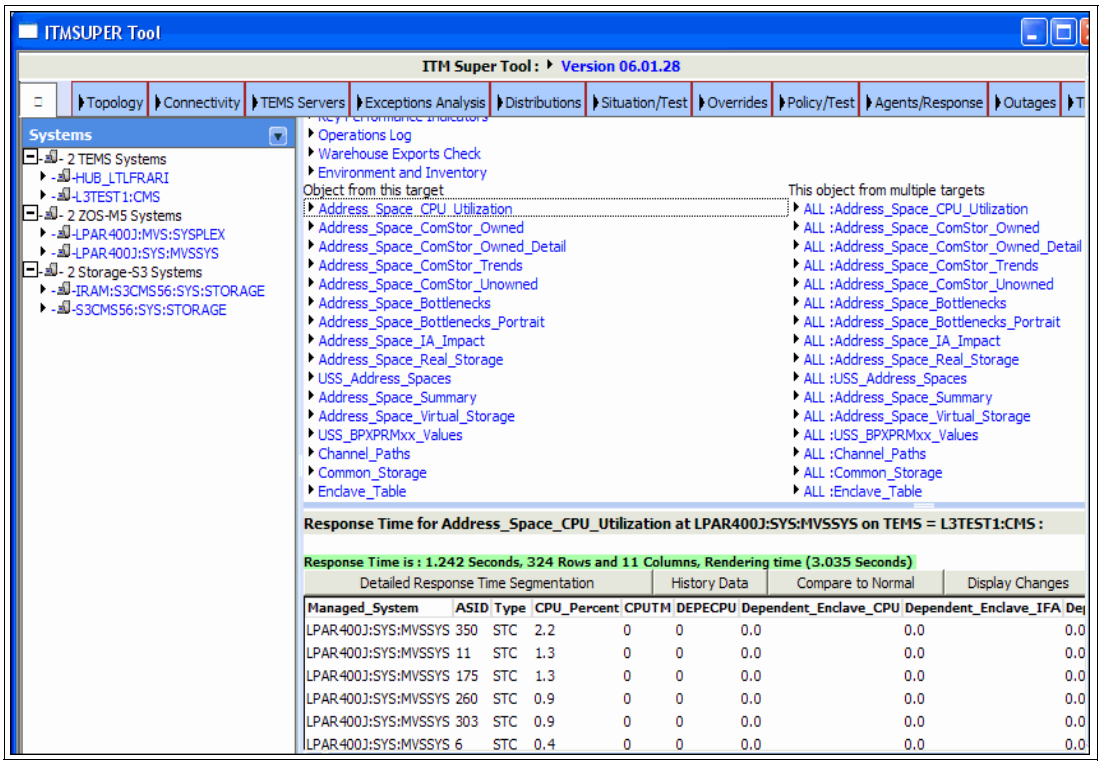


Figure B-10 ITMSuper with data for the attribute group

You can see the actual attribute name within the group being used as column headers in the output, for example ASID, Type, and CPU_Percent.

As a method for discovering the attribute group names that you need to put into the `<object>` tag of an ITM SOAP request, using this tool in this way works quite well. It is, however, subject to a couple of caveats, as follows:

- While you can see the attribute group name from the display, unless the agent returns at least one row of data for the attribute group, you will not see any actual attribute names as column heading. The impact of this is that you cannot determine what data you will see until you actually get some data back from the agent. However, in its most basic form, the attribute group name is sufficient, at least initially.
- Some products such as the Omegamon XE on z/OS product (the M5 products under the zOS-M5 systems tab in Figure B-10 on page 192) provide multiple managed system names. In this case one is for the sysplex itself and one is for an LPAR under that sysplex. The tool, however, does not distinguish between the attribute groups for the sysplex component and those for the LPAR component; everything is together in one big attribute list for the M5 product code. If you select an attribute group for the wrong component, for example Address_Space_CPU_Utilization when you have the sysplex-managed system selected, you will not see any data in the response area. Sometimes it is obvious which attribute groups belong to which managed system types for a product code, sometimes it is not so obvious and you have to experiment a little.

So now we have a method of finding out at least the attribute group names we need to plug into our SOAP request. We now need to add to modify the request to specify which managed system we want to get the data from.

Specifying the managed system name

Our original request to obtain the list of managed system names was sent to the hub TEPS. Now we need to indicate which managed system we want to obtain the requested attribute group data from so that ITM can send the request to the correct TEMA to run the appropriate agent and collect the requested data.

You do this by placing the managed system name, for example LPAR400J:SYS:MVSSYS, in the `<target>` node of the SOAP request. So now our request might look as shown in Example B-8.

Example B-8 SOAP request with managed system name

```
<CT_Get>
<userid>sysadmin</userid>
<password>mypasswd</password>
<object>Address_Space_CPU_Utilization</object>
<target>LPAR400J:SYS:MVSSYS</target>
</CT_Get>
```

This is actually enough to get, in this example, the Address Space CPU Utilization data for all address spaces on the target managed system. There are, however, a few issues with this data:

- Each ROW of data consists of 51 individual attributes for this particular Attribute group on my ITM system.
- The request returned 322 rows of data, representing the 322 currently active address spaces on the z/OS system I am requesting information from in this example.
- The data is returned in a default order, if any, that is determined by the agent.

Clearly it is not practical to try to display this volume of data on a small mobile device.

Selecting the attributes to be returned

One way to reduce the volume of data being returned by the request is to specify the specific attributes that we want to be returned by the query. As we have seen, the default action if we just specify the attribute group name in the `<target>` tag is to return everything. In order to limit the response to only a specified set of attributes from the attribute group, we can add one or more occurrences of the `<attribute>` tag to the request, each of which contains one attribute name that we want to be returned in each row of data.

For example, let us say we just want to return the job name (Job_Name attribute) and CPU percentage being used (CPU_Percent attribute) for each address space. We would modify our previous request so that it looks as shown in Example B-9.

Example B-9 Request to obtain job name and CPU percentage being used

```
<CT_Get>
<userid>sysadmin</userid>
<password>mypasswd</password>
<object>Address_Space_CPU_Utilization</object>
<target>LPAR400J:SYS:MVSSYS</target>
<attribute>Job_Name</attribute>
<attribute>CPU_Percent</attribute>
</CT_Get>
```

Now when we run the request, each row of the response only contains the selected two attributes, as shown in Example B-10.

Example B-10 Response

```
<ROW>
<Job_Name>S3CMS73</Job_Name>
<CPU_Percent dt="number">15.6</CPU_Percent>
</ROW>
```

More filtering

We still have the issue of the number of rows of data being returned, though. Remember, we want to display this data on a mobile device and therefore the aim is to only show critical or important data. For this particular attribute group (Address space CPU utilization) critical or important would probably mean address spaces with a high CPU consumption percentage.

This sort of filtering is easily achieved through the use of the `<afilter>` tag in the request. The data within the `<afilter>` tag has the form:

```
Attrbute_name; predicate; value
```

In our example, the attribute name would be CPU_Percent.

Predicate is a two-character test, for example EQ for EQual or NE for Not Equal. In our example we would use GT for Greater Than since we want all address spaces using a greater percentage of the CPU than we specify in the filter.

Let us say we want all address spaces using more than 50% of the CPU—we would code the afilter tag as shown in Example B-11

Example B-11 Filtering data

```
<afilter>CPU_Percent;GT;50<afilter>
```

When we run the request, we get the result shown in Example B-12.

Example B-12 Result

```
<ROW>
<Job_Name>S3CMS73</Job_Name>
<CPU_Percent dt="number">18.1</CPU_Percent>
</ROW>
```

Hey, wait a minute! What's going on here? We requested entries with CPU Percentage greater then 50 but we got one back that says 18.1 (%). Look at the value that is being returned. It is formatted to *one* decimal place. So you might think that you could just put 50.0 in the afilter data and that would work. This is not true. You have to input the data as an integer but to the correct number of decimal places. So if, as in this case, the output data is formatted to one decimal place, we need to enter 500 as the filter value.

If the output were formatted to two decimal places, we would need to enter 5000 in order for it to be tested as 50.00 against the data from the agent.

This means that if you want to filter numeric data in this manner you need to note how the output data is formatted and adjust the filter data to account for the correct number of decimal places. Do not worry, though, if a field is defined to display to two decimal places, it *always*

displays to two decimal places. Thus, if a returned value is for example 18.1, it would be displayed as 18.10.

You can enter multiple <afilter> filters within a query. In that case they are always ANDed together. Thus you could select, for example, address spaces with CPU usage within a range by coding as shown in Example B-13.

Example B-13 Multiple filters

```
<afilter>CPU_Percent;GE;250<afilter>
<afilter>CPU_Percent;LE;750<afilter>
```

This would select all entries with CPU_Percent values between 25.0 and 75.0.

The only time you do not need to be concerned about the number of decimal places for the data is when testing against a value of zero. In that case it is quite acceptable (and it works) to specify a single zero digit as shown in Example B-14.

Example B-14 Filtering with 0

```
<afilter>CPU_Percent;GE;0<afilter>
```

This would select all address spaces with a CPU_Percent value greater than zero.

There is no way using this method of selecting data, to specify an OR condition for the filtering. It is possible to do that within a SOAP request but the mechanism to specify the request is beyond the needs of this book and needs additional information that is not readily available.

Further reducing the data

So far we have managed to reduce the amount of data significantly, however we could still end up with more data than we want. For example, selecting all address spaces with a CPU_Percent value greater than zero could still result in a hundred or more address space rows being returned in the SOAP response.

Unfortunately, at this point you will need to revert to some programming within the server side code of your application that is making the SOAP requests to the ITM SOAP server on behalf of the mobile device.

There is no mechanism within the SOAP interface to indicate that you only want a specific number of rows or less to be returned. You will have to limit the number of rows that you will send to the mobile device within your server code. The side effect of this is that on occasion, far more data than is really necessary is being transferred between the ITM SOAP server and your server code. This is why it is important to reduce the data you are requesting to the absolute minimum set of attributes (using attribute tags) and range of data required to address the business problem (using afilter tags) to reduce the overhead associated with the network transfer of large amounts of data between the ITM SOAP server and your server code.

Sorting the data

Again, there is no facility within this mechanism to have ITM sort the data before it is returned to the requestor. It is possible, but not with this method of obtaining the data. Again, that method is beyond the scope of this book.

This means that your server side code must also include some mechanism to sort the data into the required order before the number of rows is reduced as described above.

While you can implement client side sorting on the mobile device, it is only going to be able to sort the data it receives. If that data is trimmed before sorting the users may not actually get the data that they expect.

C

Deployment to WebSphere Application Server on z/OS

Some of the scenarios discussed in this book have a middle-tier component that runs in WebSphere Application Server. If this is the case, this component can also be deployed to WebSphere Application Server on z/OS or on Linux for System z. In this appendix we provide the common steps to follow when you wish to deploy the WebSphere Application Server component to z/OS or Linux for System z.

© Copyright IBM Corp. 2010. All rights reserved.

Establishing a connection to a remote WebSphere Application Server

The Rational IDEs can help you to test, debug and publish or deploy applications on local test environments as well as remote servers. To test the application locally you can install the WebSphere test environment and make a connection to a local test environment from the Server view.

In this section, however, we are explaining the steps to connect to a remote WebSphere Application Server for application testing or deployment purposes. Before we start with these steps, make sure that your remote WebSphere Application Server is running.

1. Right-click the **Servers** view and select **New** → **Server** from the pop-up context menu.
2. Specify the Hostname or IP address of the remote system in the Server's host name text field and select the WebSphere Application Server version that you are using. In our case we were using WebSphere Application Server v7.0, as shown in Figure C-1. Click **Next >**.

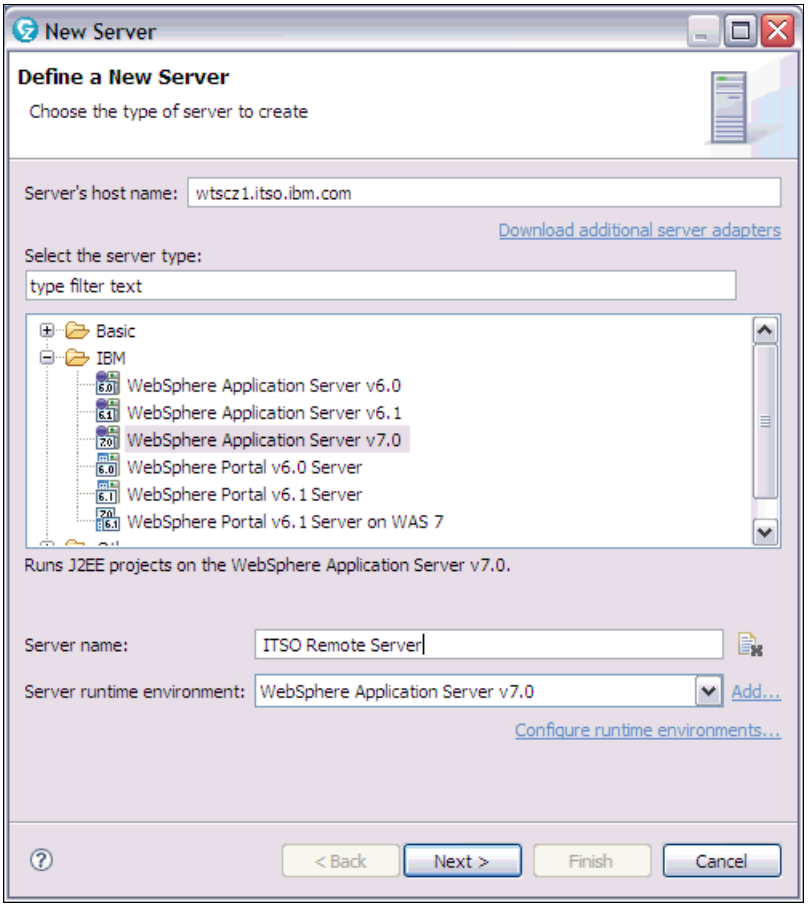


Figure C-1 Define a new server - select an application server version

3. In the window that follows, update the port numbers for the RMI and SOAP protocols. You can get these port number values in use from your WebSphere Application Server administrator or find them by logging into the Integrated Solutions Console of the remote application server. If security is enabled on the server, then you also need to select the checkbox **Security is enabled on server** and enter the user ID and Password, as shown in Figure C-2. Click **Finish.**

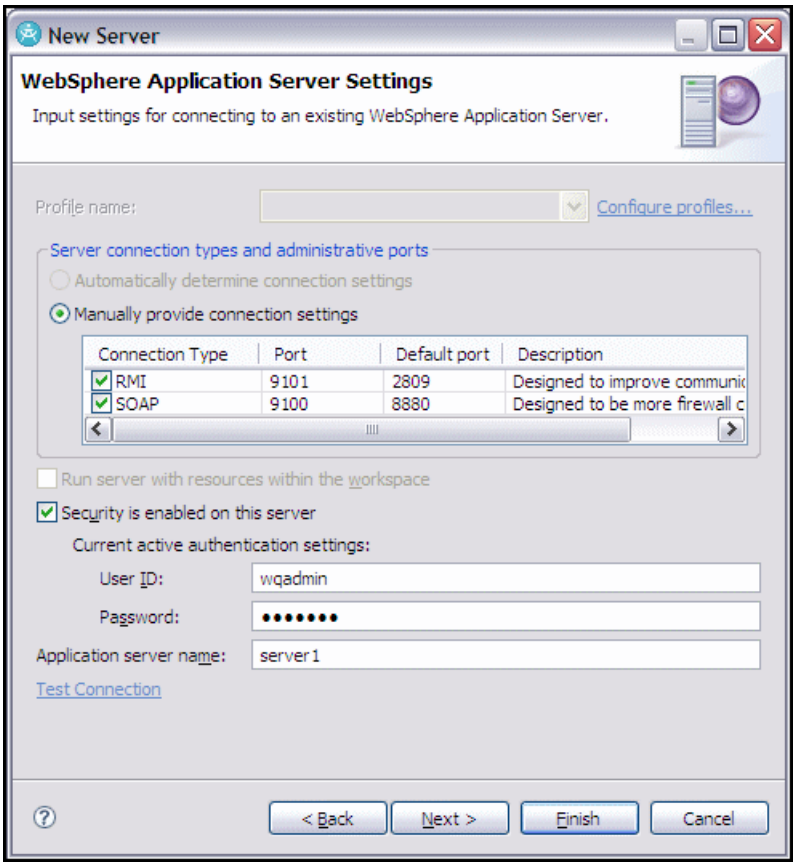


Figure C-2 New Server definition - define port and security settings

4. Once you finish with the new server wizard, you can see the status of the server in the Servers view (Figure C-3). You can see that the server status is “started”. We have established a connection to the remote server for application deployment and testing purposes.

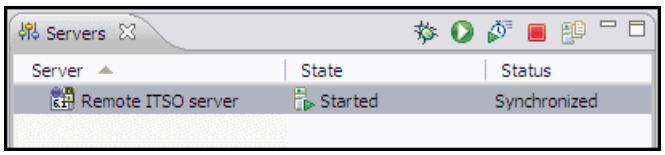


Figure C-3 Server status

5. In case you want to update a setting of the server connection configuration, you can double-click the server in the Servers view, which will result in opening an overview page. The server connection configuration information can be updated here. (See Figure C-4 on page 200).

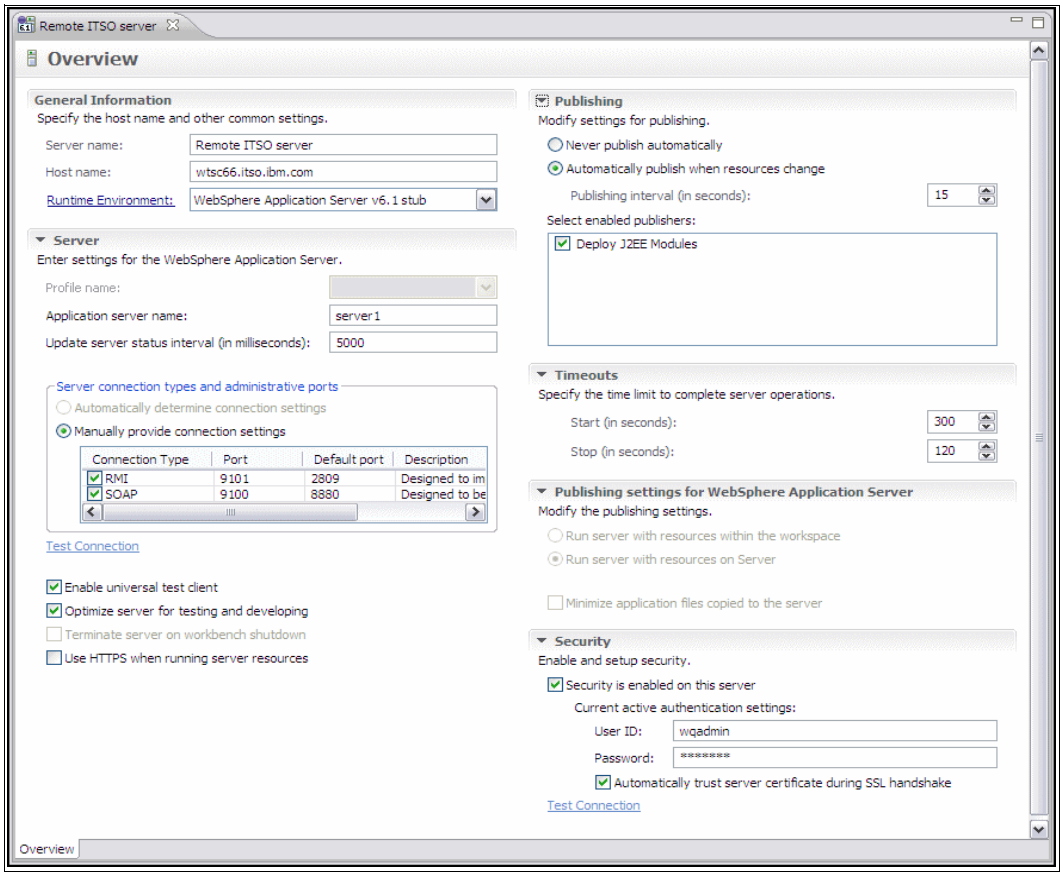


Figure C-4 Server connection configuration

6. To run the administrative console of the remote application server, right-click the server in the Servers view and select **Administration → Run administration console** (Figure C-5).

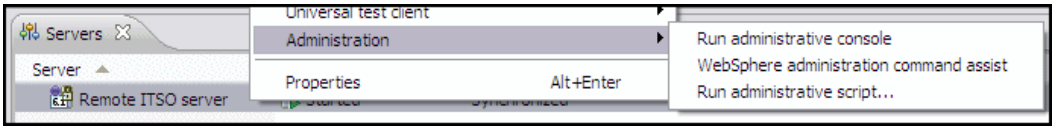


Figure C-5 Server - Launch Integrated Solutions Console

7. You will now see the Integrated Solutions Console opened in RDz and, if security is enabled at the remote WebSphere Application Server, it will prompt you to insert your User ID and Password. See Figure C-6 on page 200.

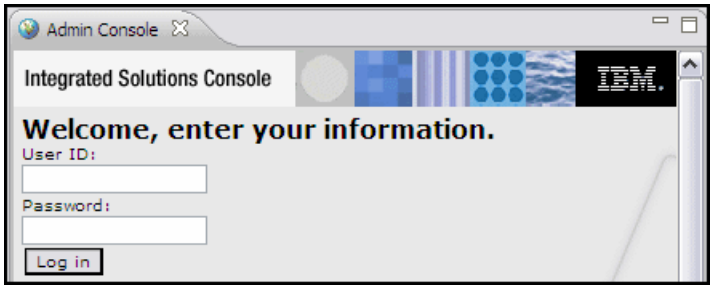


Figure C-6 Server - Integrated Solutions Console log in

8. After establishing the connection successfully to the remote server, you can perform various administrative tasks on the server, provided you have admin access.

After being connected to a remote server, you can follow the same steps as you would follow before deploying to a server on the local Windows system. The only difference is that you deploy your middle-tier component to a remote server instead of a local server.

In our case we have been deploying application samples to a server with hostname wtscz1.itso.ibm.com.

Once you have deployed the application to a remote server, you can point your smartphone browser to the address (of the remote server) and start testing your application.

D

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG247836

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247836.

Using the Web material

The additional Web material that accompanies this book includes the following files.

Accessing z/OS applications with HATS from Apple iPhone

File name	*Description*
HATS_EAR6.ear	EAR file of the project containing the HATS example discussed in "Accessing z/OS applications with HATS from Apple iPhone" on page 21.
HATS_EAR6.zip	Project Interchange File containing the HATS example discussed in "Accessing z/OS applications with HATS from Apple iPhone" on page 21.

© Copyright IBM Corp. 2010. All rights reserved.

Accessing mainframe resources from Android mobile device applications

File name	*Description*
Chapter 5.zip	All sample code for the scenarios described in Chapter 5, “Accessing mainframe resources from Android mobile device applications” on page 73. This .zip file contains two .zip files: ► A .zip file with the VSAM example folder containing the source code of the examples which are described in “Accessing mainframe resources from native smartphone applications” on page 77. ► A .zip file with the push example folder containing the source code of the example which is described in “Pushing information to smartphones” on page 83. Each folder contains a README.txt file that describes detailed steps for setting up the development environment as well as configuring the execution environment.

Accessing a CICS Web service with an EGL Rich UI client on Apple iPhone

File name	*Description*
iPhoneEGL.zip	Project Interchange File containing the EGL Rich UI example discussed in “Accessing a CICS Web service with an EGL Rich UI client on Apple iPhone” on page 93.

Accessing a CICS Restful Web service from Apple iPhone

File name	*Description*
CicsRESTIEAR.ear	EAR file of the project containing the example discussed in “Accessing z/OS applications with HATS from Apple iPhone” on page 21.
CicsRESTIEAR.zip	Project Interchange File containing the example discussed in “Accessing z/OS applications with HATS from Apple iPhone” on page 21.

Accessing IBM Tivoli Monitoring from smartphones

File name	*Description*
ITMClient.ear	EAR file of the project containing the ITM example discussed in “Accessing IBM Tivoli Monitoring from smartphones” on page 151.
ITMClient.zip	Project Interchange File containing the ITM example discussed in “Accessing IBM Tivoli Monitoring from smartphones” on page 151.

How to use the Web material

To be able to use the additional material successfully, you first need to make sure that you have the prerequisite tools installed for each specific scenario. You can find this information in each chapter.

Examples that use Rational IDE tooling

For the examples that use a Rational IDE, the safest is to import the Project Interchange File, as this will restore all workbench settings as well. Importing an EAR file will only restore the project-specific artifacts.

Take the following steps, once you have the prerequisite tools installed:

1. Download the desired EAR or ZIP file into a local directory on your PC.
2. Start up the required tool, e.g. Rational Developer for System, Rational Business Developer or Rational Application Developer.

   For the HATS example you will need the HATS Toolkit installed as well.
3. Select **File** → **Import**, and select the file type you plan to import.
4. Browse to the directory where you downloaded the EAR or ZIP file, and import the file.

Android examples

The ZIP file containing the Android examples contains two README files explaining how to use the additional material.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 208. Note that some of the documents referenced here may be available in softcopy only.

- *Application Development for CICS Web Services,* SG24-7126
- *Implementing CICS Web Services,* SG24-7206
- *Securing Access to CICS Within an SOA*, SG24-5756
- *Enabling z/OS Application for SOA,* SG24-7669.
- *Securing CICS Web Services*, SG24-7658
- *Powering SOA Solutions with IMS*, SG24-7662
- *Integrating Backend Systems with WAS on z/OS through Web Services*, SG24-7548
- *Websphere Application Server 7.0 Security Guide*, SG24-7660
- *Websphere Application Server 7.0 Web Services Guide*, SG24-7758
- *Building Dynamic Ajax Applications with WAS FP for Web 2.0*, SG24-7635
- *WebSphere MQ V6 Fundamentals*, SG24-7128

Other publications

These publications are also relevant as further information sources:

- *IMS SOAP Gateway User’s Guide and Reference,* SC19-1290
- *IMS Version 10 Application Programming Guide*, SC18-9698

Online resources

These Web sites are also relevant as further information sources:

- Web 2.0 Applications with EGL Rich UI
  `http://www-949.ibm.com/software/rational/cafe/docs/DOC-2868`
- Android and iPhone Browser Wars Part 1- WebKit to the rescue
  `http://www.ibm.com/developerworks/opensource/library/os-androidiphone1/`
- Android and iPhone Browser Wars Part 2 - Build browser-based applications
  `http://www.ibm.com/developerworks/opensource/library/os-androidiphone2/index.html`
- Developer iPhone Web applications with Eclipse
  `http://www.ibm.com/developerworks/opensource/library/os-eclipse-iphone/`

© Copyright IBM Corp. 2010. All rights reserved.

- Modern Application Development featuring Web 2.0 for system z
  http://www.ibm.com/developerworks/offers/techbriefings/details/appdevwebz.html
- EGL is ready to help you adopt Web 2.0
  http://www.ibm.com/developerworks/websphere/techjournal/0807_col_barosa/0807_col_barosa.html
- Enterprise Generation Language Resources
  http://www.ibm.com/developerworks/rational/products/egl/egldoc.html
- EGL Rich User Interface Hub
  http://www-949.ibm.com/software/rational/cafe/community/egl/rui
- Android Developer's Guide
  http://developer.android.com/guide/basics/what-is-android.html
- Building iPhone applications (for developers)
  http://building-iphone-apps.labs.oreilly.com/index.html

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

© Copyright IBM Corp. 2010. All rights reserved.

I

J

L

M

P

R

S

T

U

V

W

X

IBM Redbooks

System z on the Go: Access to z/OS from Smartphones

(0.2”spine)
0.17”<->0.473”
90<->249 pages

System z on the Go Access to z/OS from Smartphones

Sample scenarios to get started with Apple iPhone and Android

Solution architectures and modernization considerations

Step-by-step guide for Web and native applications

In this IBM Redbooks publication we demonstrate that it is possible to combine the traditional strengths of the mainframe to manage large volumes of data and run business transactions with the Web 2.0 paradigm. We can get simpler interfaces, better integration among different services, lightweight protocols for communication, and much more, together with the availability, security, and reliability of mainframe data. And we will show how mainframe data can be accessed by smartphones such as Android or iPhone.

But we can do more to demonstrate how flexible the mainframe platform is. Through the use of pervasive devices it is possible to add new possibilities to mainframe applications, extending System z capabilities. We can receive notifications in real time, for example, of successful or unsuccessful termination of a TWS job stream, or we can immediately get alerts about abends that occurred in a critical application.

This book is another demonstration that the mainframe is alive and kicking and can and should play a key role in modern application architectures.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7836-00 ISBN 0738434434